A

PROJECT REPORT ON

Machine Learning-Based Spam Comments Detection On Youtube

Submitted in partial fulfillment of the requirements for the award of the degree of

MASTER OF COMPUTER APPLICATIONS

By

PILLI VENKATA DHARANI

(23BFF00097)

Under the esteemed guidance of

Mr. M. VASU
Assistant Professor
Department of MCA

DEPARTMENT OF MASTER OF COMPUTER APPLICATIONS

SRI VENKATESWARA COLLEGE OF ENGINEERING (AUTONOMOUS)

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapur)
Accredited by NAAC with 'A' Grade
Opp. LIC Training Centre, Karakambadi Road, TIRUPATI–517507

2023-2025

***

SRI VENKATESWARA COLLEGE OF ENGINEERING (AUTONOMOUS)

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapur)

Accredited by NAAC with 'A' Grade

Opp. LIC Training Centre, Karakambadi Road, TIRUPATI – 517507

DEPARTMENT OF MASTER OF COMPUTER APPLICATIONS

2023-2025

CERTIFICATE

This is to certify that the project report entitled ***"Machine Learning Based Spam Comments Detection On YouTube"*** is a bonafide record of the project work done and submitted by

PILLI VENKATA DHARANI

(23BFF00097)

for the partial fulfillment of the requirements for the award of **MASTER OF COMPUTER APPLICATIONS** Degree from SRI VENKATESWARA COLLEGE OF ENGINEERING (AUTONOMOUS) Affiliated to JNT University Anantapur.

GUIDE **HEAD OF THE DEPARTMENT**

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

I am thankful to my guide **Mr.M.VASU, Assistant Professor,** for his valuable guidance and encouragement. His helping attitude and suggestions have helped me in the successful completion of the project**.**

I would like to express my sincere thanks to **Dr. E. Sreedevi, Professor, Head of the Department of MCA,** for her kind help and encouragement during the course of my study and in the successful completion of the project work.

I express my sincere thanks to **Dr. N. Sudhakar Reddy**, **Principal,** S.V College of Engineering, Tirupati.

Successful completion of any project cannot be done without proper support and encouragement. My sincere thanks to the Management for providing all the necessary facilities during the Course of my study.

I would like to thank my parents and friends, who have the greatest contributions in all my achievements, for the great care and blessings in making me successful in all my endeavors.

I would like to express my deep gratitude to all those who helped me directly or indirectly to transform an idea into my working project.

PILLI VENKATA DHARANI

(23BFF00097)

DECLARATION

I hereby declare that project entitled **"MACHINE LEARNING BASED SPAM COMMENTS DETECTION ON YOUTUBE"** submitted to the Department of COMPUTER APPLICATIONS, **SRI VENKATESWARA COLLEGE OF ENGINEERING, TIRUPATI** in partial fulfillment of Requirement for the award of the degree of **MASTER OF COMPUTER APPLICATIONS.**

This project is the result of my own effort and it has not been submitted to any other University or Institution for the award of any degree other than specified above.

SIGNATURE OF THE STUDENT

ABSTRACT

The YouTube Spam Comment Identification project aims to classify YouTube comments as spam or not spam, addressing violations such as self-promotion, phishing, repetitive comments, and link spam. Utilizing a dataset of 1956 comments, the project employs text preprocessing and feature extraction with HashingVectorizer to prepare data for classification. Seven machine learning algorithms are implemented: Multinomial Naive Bayes, Decision Tree, AdaBoost, MLP Classifier, Support Vector Machine (SVM), K-Nearest Neighbors (KNN), and Feed-Forward Neural Network (FFNN). AdaBoost achieves the highest accuracy of 95.67%, while SVM and FFNN perform poorly due to suboptimal configurations. The project includes a Flask-based web application with MySQL integration for user authentication and comment prediction using the AdaBoost model. The system provides a practical solution for identifying spam comments, with potential for enhanced moderation on YouTube platforms.

TABLE OF CONTENTS

LISTOF FIGURES

1. INTRODUCTION

1.1 General

YouTube, as one of the largest video-sharing platforms, hosts millions of comments daily, many of which violate community guidelines through spam content such as self-promotion, phishing, repetitive comments, or malicious links. Identifying and filtering these spam comments is crucial for maintaining a safe and engaging user experience. The YouTube Spam Comment Identification project addresses this challenge by leveraging machine learning techniques to classify comments as spam or not spam, enabling automated moderation to enhance platform integrity.

1.2 Objective of the Project

The primary objective of this project is to develop a robust classification system that accurately distinguishes between spam and non-spam YouTube comments. Spam comments include self-promotion, advertising, repetitive content, misleading clickbait, phishing attempts, fake engagement, harassment, or link spam. The system aims to achieve high accuracy using machine learning algorithms and provide a user-friendly web interface for real-time comment classification, facilitating effective content moderation.

1.3 Description of the Project

The project involves a binary classification task using a dataset of 1956 YouTube comments labeled as spam (Class 1) or not spam (Class 0). The dataset is preprocessed by cleaning comment text and extracting features using HashingVectorizer. Seven machine learning algorithms are implemented: Multinomial Naive Bayes, Decision Tree, AdaBoost, MLP Classifier, Support Vector Machine (SVM), K-Nearest Neighbors (KNN), and Feed-Forward Neural Network (FFNN). The models are trained and evaluated on metrics such as accuracy, precision, recall, and F1-score. A Flask-based web application, integrated with a MySQL database for user authentication, allows users to register, log in, view the dataset, check model performance, and predict whether a comment is spam. The AdaBoost model, with the highest accuracy of 95.67%, is used for predictions in the deployed system.

.

2. LITERATURE SURVEY

The YouTube Spam Comment Identification project builds on existing research in text classification and spam detection, particularly in the context of social media platforms like YouTube. The following subsections review relevant studies and methodologies that inform this project, focusing on spam comment characteristics, dataset usage, preprocessing techniques, classification algorithms, and deployment strategies.

2.1 Characteristics of YouTube Spam Comments

Research highlights that YouTube spam comments often include self-promotion, phishing links, repetitive text, and inappropriate content, as noted in studies analyzing social media spam. These comments deviate from genuine user engagement, necessitating automated detection to maintain platform integrity. The project aligns with this by defining spam as comments violating YouTube's guidelines, such as advertising, misleading clickbait, and harassment, ensuring the classification system targets these specific traits.

2.2 Datasets for Spam Comment Classification

Previous work, such as studies on YouTube comment datasets, emphasizes the importance of labeled datasets for training classification models. The dataset used in this project, containing 1,956 comments labeled as spam (1) or not spam (0), mirrors datasets like the YouTube Spam Collection, which includes comments from popular videos. These datasets provide a balanced representation of spam and non-spam comments, enabling robust model training and evaluation, as adopted in this project.

2.3 Text Preprocessing Techniques

Text preprocessing is critical for effective spam detection, as outlined in research on natural language processing (NLP). Common techniques include converting text to lowercase, removing special characters, non-ASCII characters, and extra whitespaces, and applying feature extraction methods like HashingVectorizer. This project adopts similar preprocessing steps to clean the comment data, ensuring consistency and reducing noise, which aligns with best practices in text classification literature.

2.4 Classification Algorithms for Spam Detection

Literature on spam detection compares various machine learning algorithms, including Support Vector Machines (SVM), K-Nearest Neighbors (KNN), Neural Networks, Naive Bayes, Decision Trees, and ensemble methods like AdaBoost. Studies indicate that ensemble methods often outperform single classifiers in text classification tasks due to their ability to handle imbalanced data. This project evaluates both existing (SVM, KNN, FFNN) and proposed (Multinomial Naive Bayes, Decision Tree, MLP Classifier, AdaBoost) algorithms, drawing on these findings to identify the most effective model, with AdaBoost achieving the highest accuracy of 95.67%.

2.5 Deployment of Spam Detection Systems

Deploying machine learning models via web applications is a growing trend in applied research, as seen in studies on real-time spam filtering tools. Flask-based applications, integrated with databases like MySQL, provide user-friendly interfaces for model interaction. This project’s deployment of a Flask web application for user registration, login, dataset viewing, and real-time spam prediction using the AdaBoost model reflects these advancements, ensuring practical usability for detecting spam comments in a scalable manner.

3. SYSTEM ANALYSIS

3.1 Existing System

The existing system for YouTube spam comment identification relies on traditional machine learning algorithms, specifically Support Vector Machine (SVM) and K-Nearest Neighbors (KNN). These methods classify comments as spam or not spam based on textual features extracted from comment content. SVM uses a hyperplane to separate classes in a high-dimensional space, while KNN classifies comments by comparing them to the nearest labeled examples in the feature space. The existing system processes comment text and evaluates performance using metrics such as accuracy, precision, recall, and F1-score.

Disadvantages of Existing System

- **Poor Performance of SVM**: The SVM implementation in the project achieves a low accuracy of 46.67%, with precision, recall, and F1-score of 48%, 47%, and 30%, respectively, due to suboptimal hyperparameters (e.g., low C=0.01 and gamma=0.001).
- **KNN Limitations**: KNN yields a moderate accuracy of 86.33% but struggles with high-dimensional sparse data, leading to lower precision (87%) and recall (86%) compared to other methods.
- **Computational Inefficiency**: Both SVM and KNN are computationally intensive for large datasets, making them less scalable for real-time spam detection on platforms like YouTube.
- **Limited Feature Handling**: The existing system relies solely on HashingVectorizer for feature extraction, which may introduce feature collisions and fail to capture complex linguistic patterns.

3.2 Proposed System

The proposed system enhances YouTube spam comment identification by implementing a suite of advanced machine learning algorithms: Multinomial Naive Bayes, Decision Tree, AdaBoost, MLP Classifier, and Feed-Forward Neural Network (FFNN), alongside the existing SVM and KNN. The system uses a dataset of 1956 comments, preprocessed with text cleaning and feature extraction via HashingVectorizer. The models are trained on 700 samples and tested on 300, with AdaBoost achieving the highest accuracy of 95.67%. A Flask-based web application, integrated with a MySQL database, enables user registration, login, dataset viewing, model performance comparison, and real-time comment prediction using the AdaBoost model.

Advantages of Proposed System

- **Higher Accuracy**: AdaBoost achieves 95.67% accuracy, with precision, recall, and F1-score of 95.76%, 95.67%, and 95.67%, respectively, outperforming the existing SVM and KNN.
- **Diverse Algorithms**: The inclusion of Multinomial Naive Bayes (91.33% accuracy), Decision Tree (94.33%), and MLP Classifier (92.33%) provides robust classification options, leveraging different strengths such as probabilistic modeling and ensemble learning.
- **User-Friendly Interface**: The Flask web application offers an intuitive platform for users to interact with the system, including registration, login, and comment prediction.
- **Scalable Deployment**: Model persistence using pickle and h5 formats allows for efficient reuse, and the MySQL database supports user management for scalable access.
- **Effective Preprocessing**: Text cleaning removes noise (e.g., special characters, non-ASCII text), and HashingVectorizer ensures memory-efficient feature extraction for text data.

3.3Hardware & Software Requirements

3.3.1 Hardware Requirements

- **Processor**: Intel Core i3 or higher (or equivalent).
- **RAM**: Minimum 4 GB (8 GB recommended for model training and web application).
- **Storage**: At least 500 MB free disk space for dataset, models, and MySQL database.
- **Internet Connection**: Required for Flask web application deployment and testing.

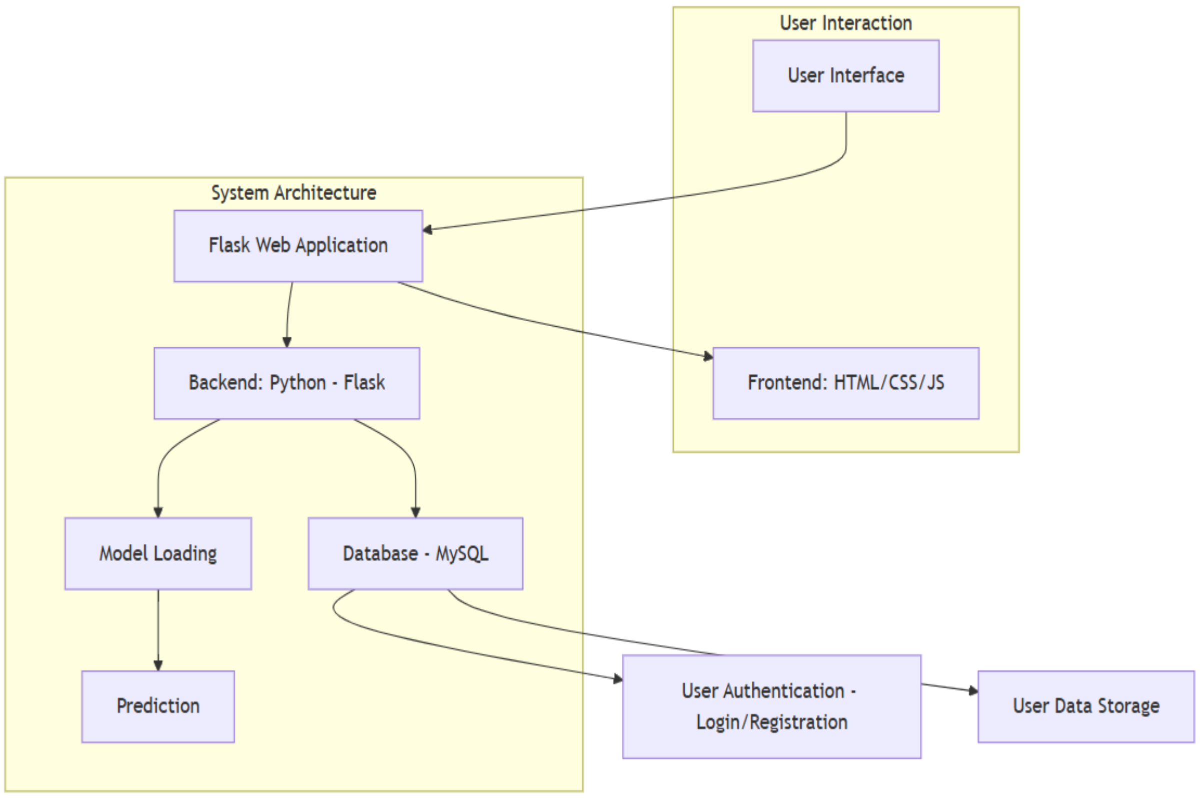
3.3.2 Software Requirements

- **Operating System**: Windows
- **Programming Language**: Python 3.10.8.
- **Libraries and Frameworks**:
  - **Data Processing**: pandas, numpy.
  - **Machine Learning**: scikit-learn (for SVM, KNN, Naive Bayes, Decision Tree, AdaBoost, MLP Classifier), tensorflow (for FFNN).
  - **Feature Extraction**: scikit-learn's HashingVectorizer.
  - **Visualization**: matplotlib, seaborn.
  - **Web Development**: Flask.
  - **Database**: MySQL (version 8.0 or higher).
- **Development Environment**: Jupyter Notebook for model building (CODE.ipynb).
- **Database Management**: MySQL Workbench or similar for managing the youtube database.
- **Web Browser**: Chrome, Firefox, or Edge for accessing the Flask application.

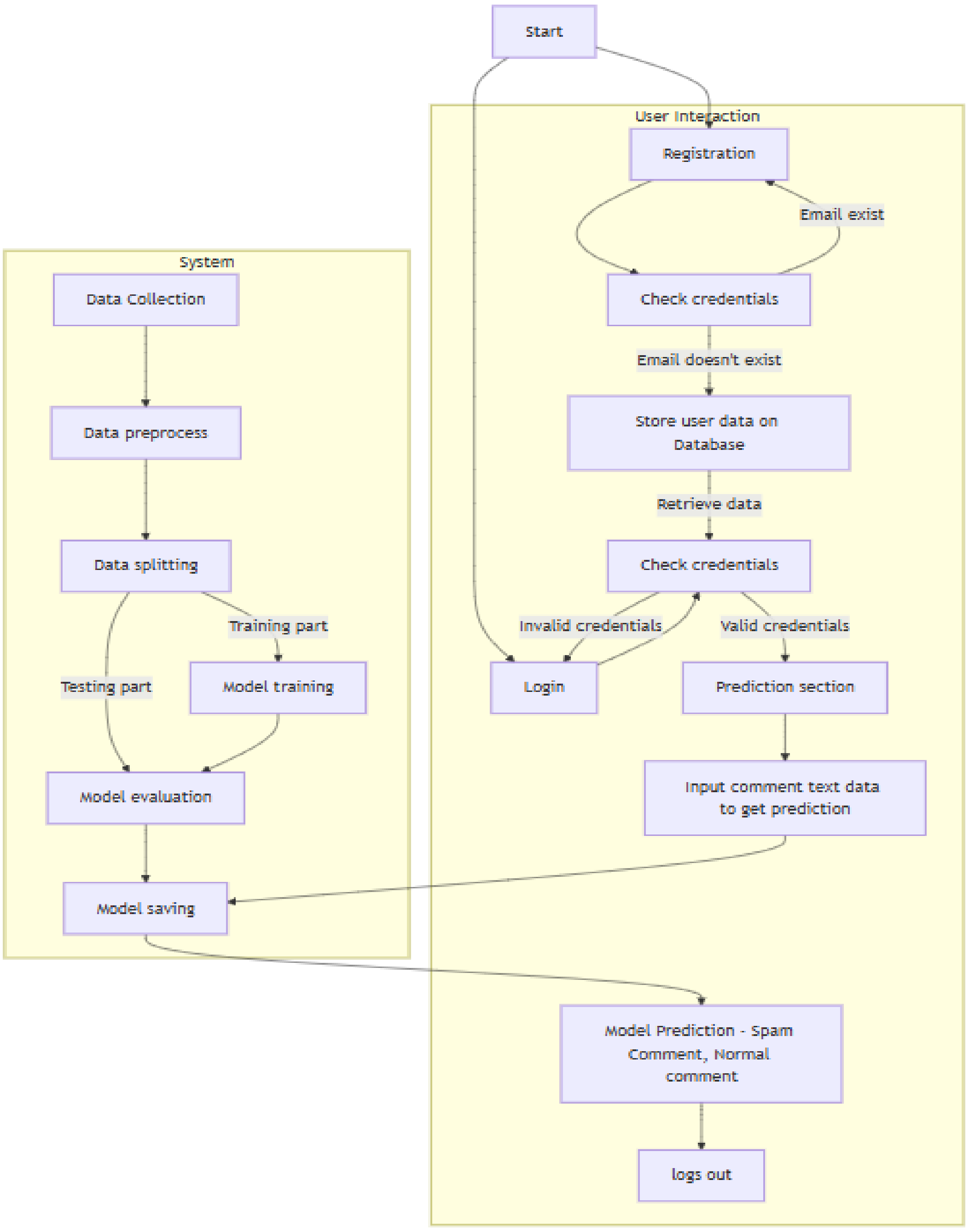
.

4. SYSTEM DESIGN

4.1 SYSTEM ARCHITECTURE:



Project work flow diagram



4.2 UML DIAGRAMS

UML stands for Unified Modelling Language. UML is a standardized general-purpose modelling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group.

The goal is for UML to become a common language for creating models of object-oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

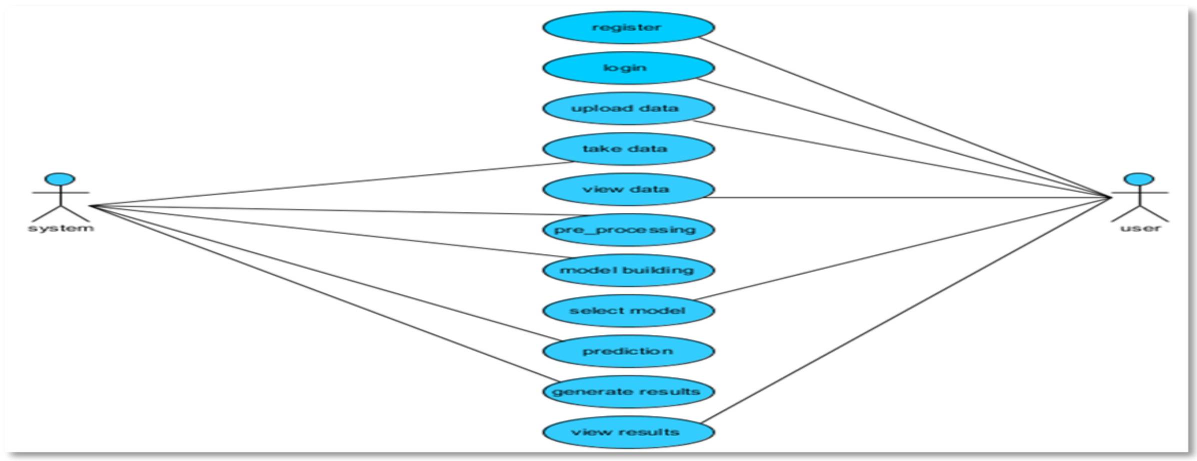
The Unified Modelling Language is a standard language for specifying, Visualization, Constructing and documenting the artefacts of software system, as well as for business modelling and other non-software systems.

The UML represents a collection of best engineering practices that have proven successful in the modelling of large and complex systems.

The UML is a very important part of developing objects-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

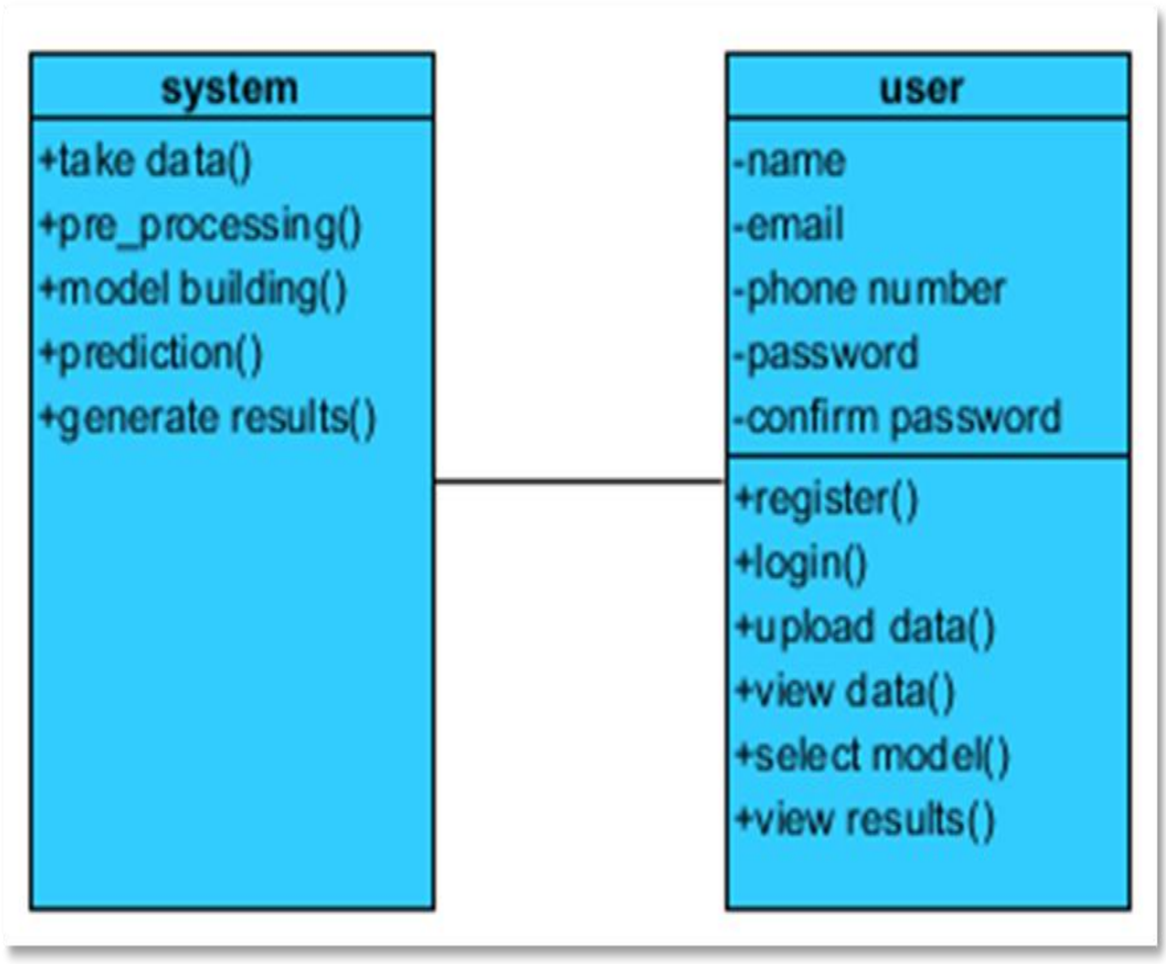
USE CASE DIAGRAM

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases.



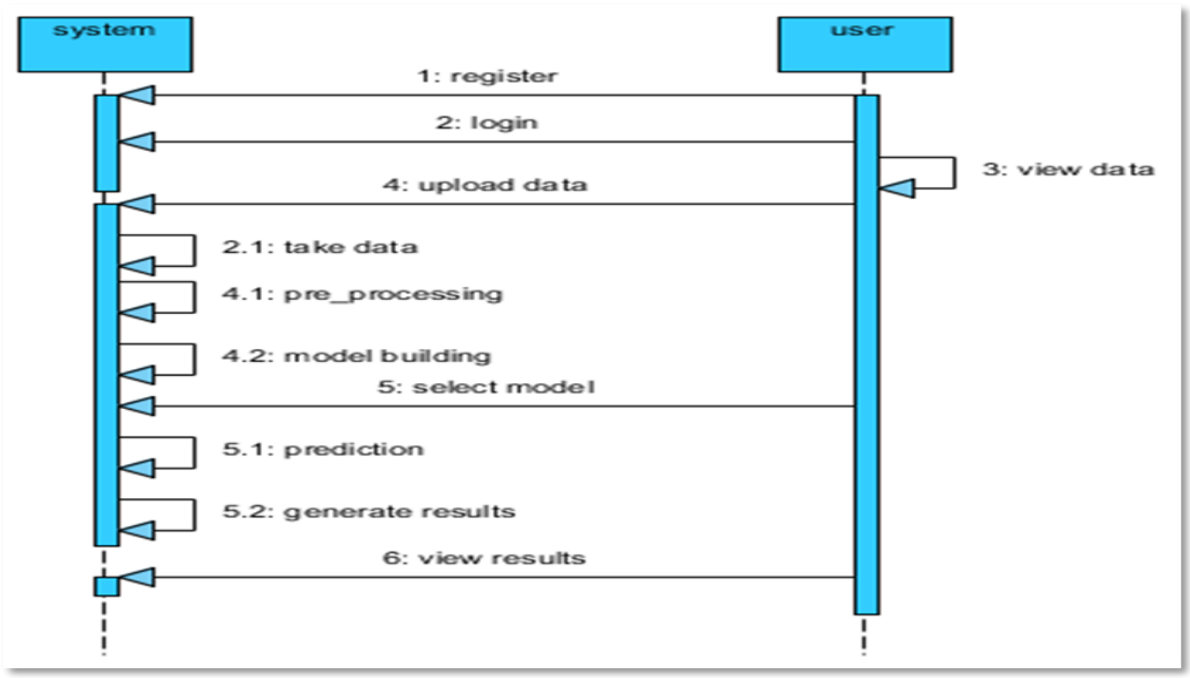
CLASS DIAGRAM

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. It explains which class contains information



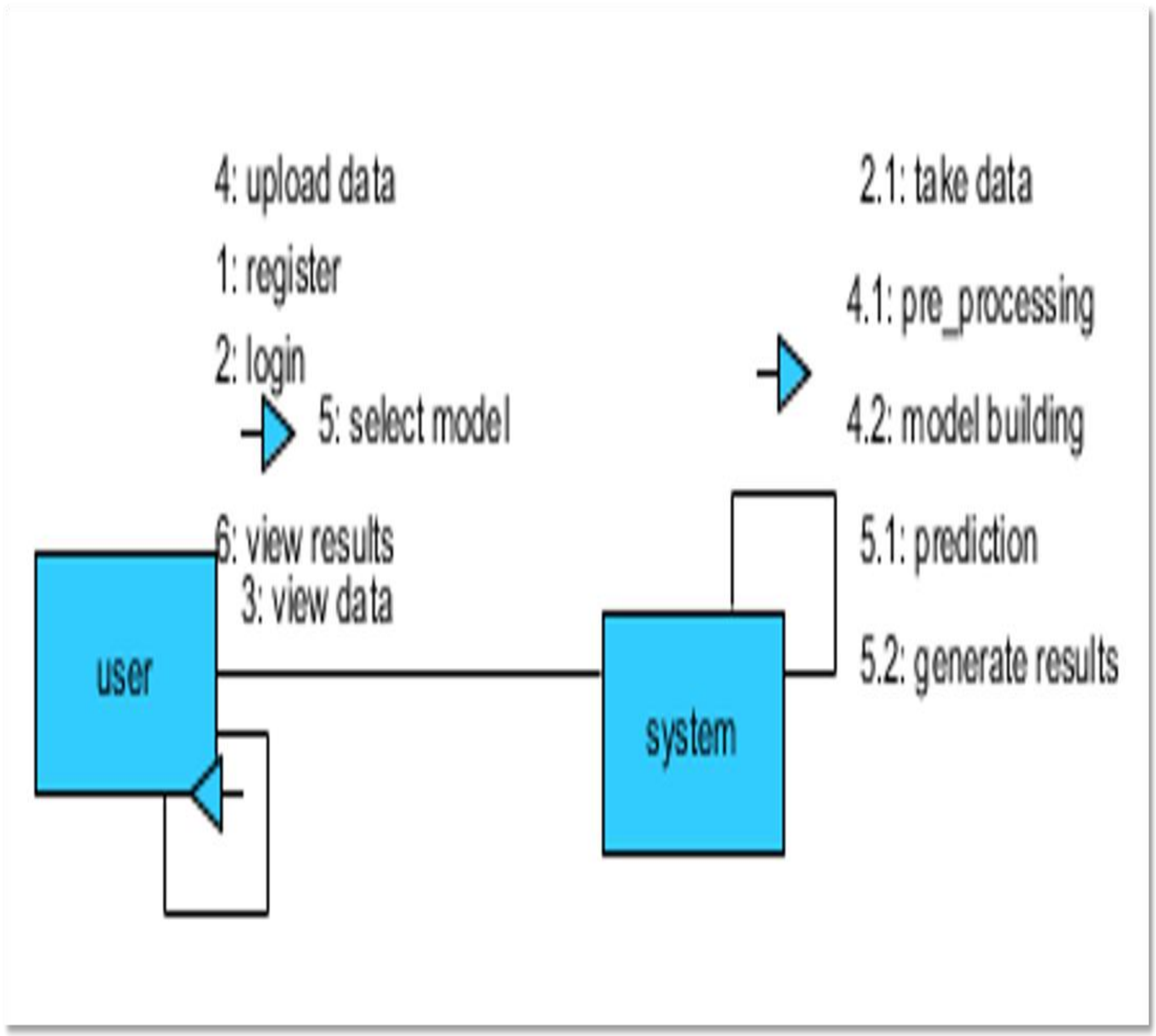
SEQUENCE DIAGRAM

- A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order.
- It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams



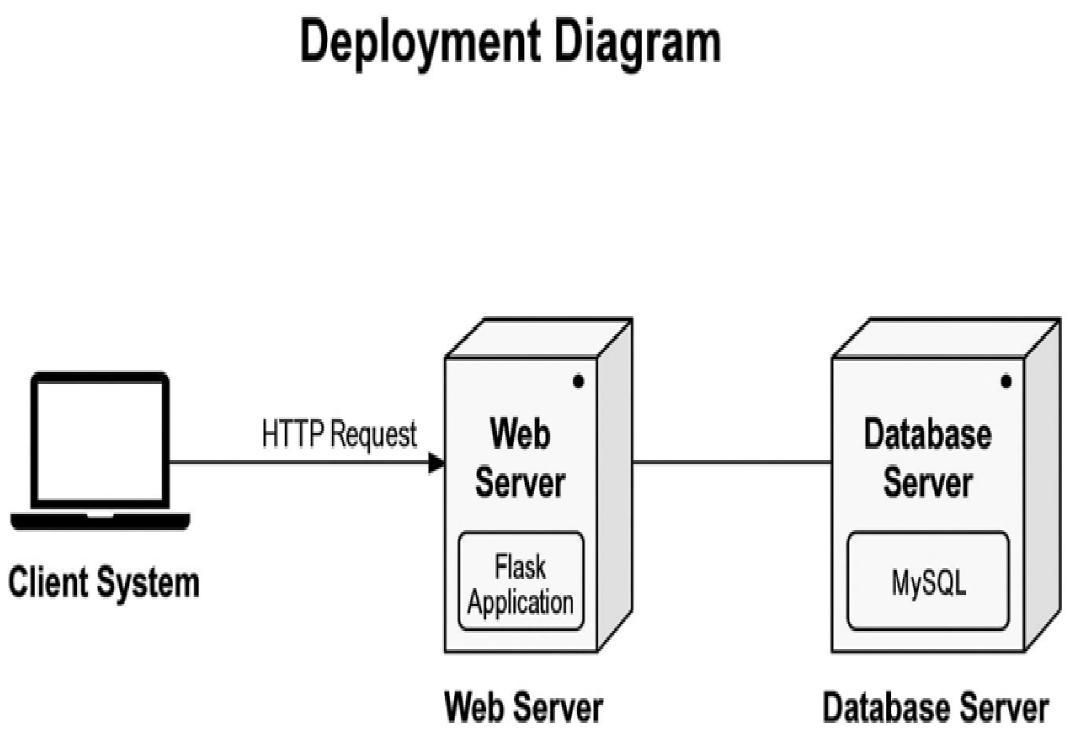
COLLABORATION DIAGRAM:

In collaboration diagram the method call sequence is indicated by some numbering technique as shown below. The number indicates how the methods are called one after another. We have taken the same order management system to describe the collaboration diagram. The method calls are similar to that of a sequence diagram. But the difference is that the sequence diagram does not describe the object organization whereas the collaboration diagram shows the object organization.



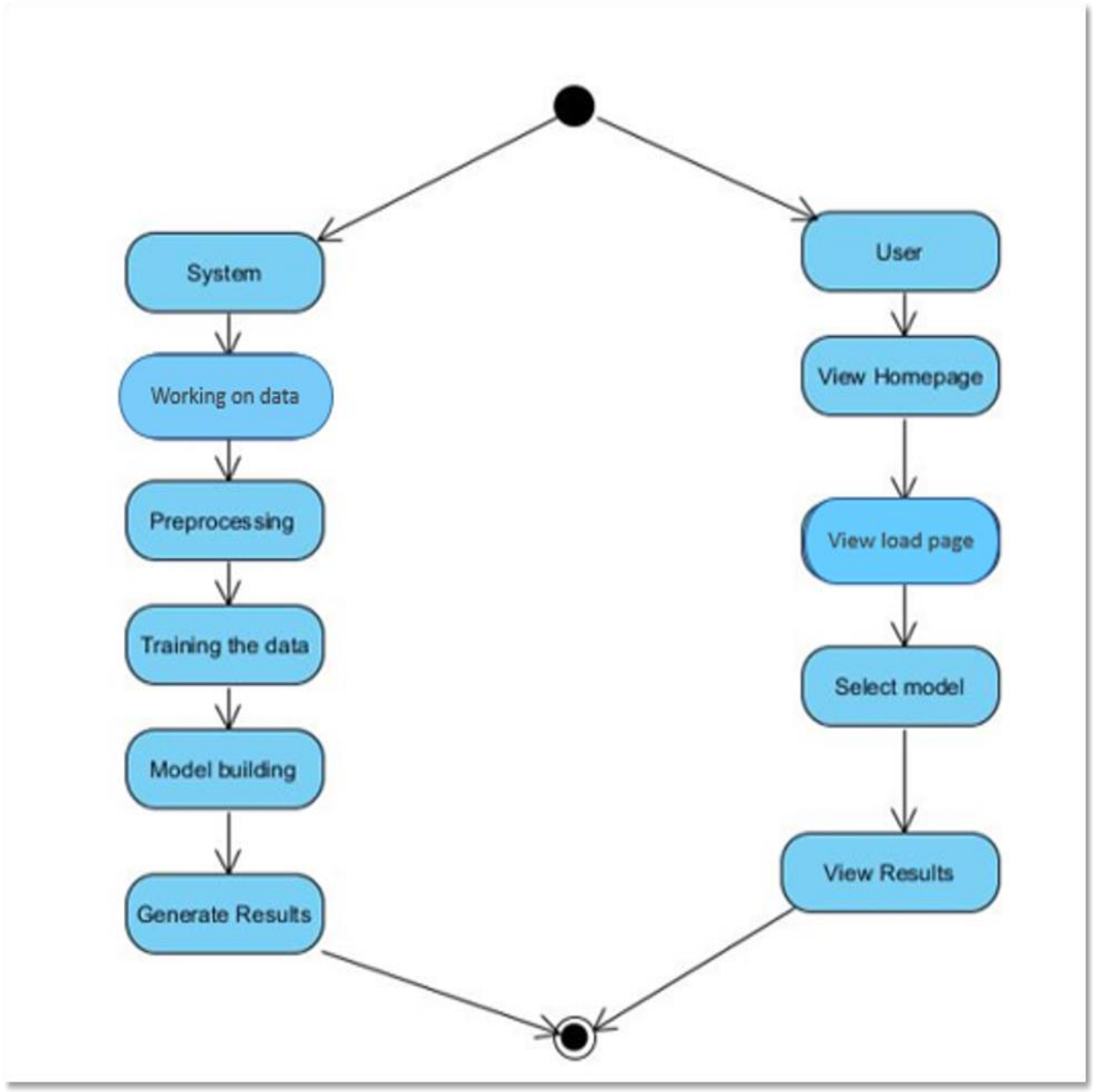
DEPLOYMENT DIAGRAM

Deployment diagram represents the deployment view of a system. It is related to the component diagram. Because the components are deployed using the deployment diagrams. A deployment diagram consists of nodes. Nodes are nothing but physical hardware's used to deploy the application.



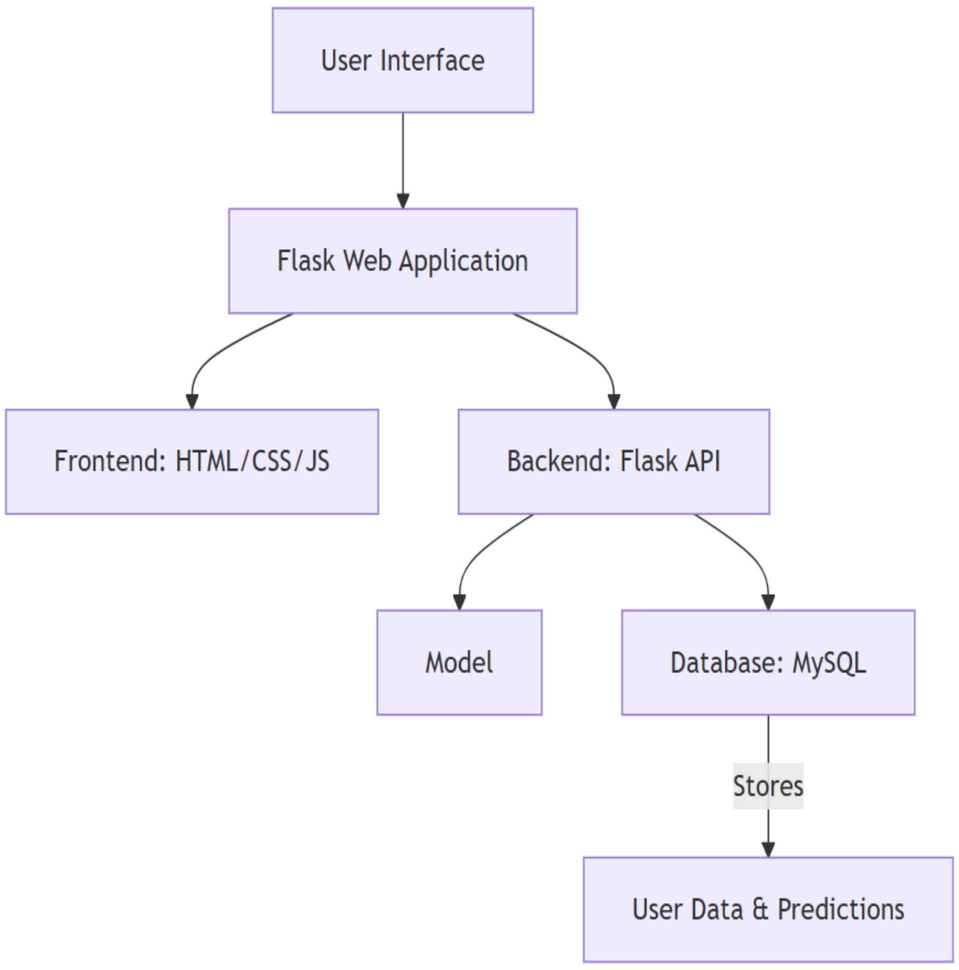
ACTIVITY DIAGRAM:

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modelling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.



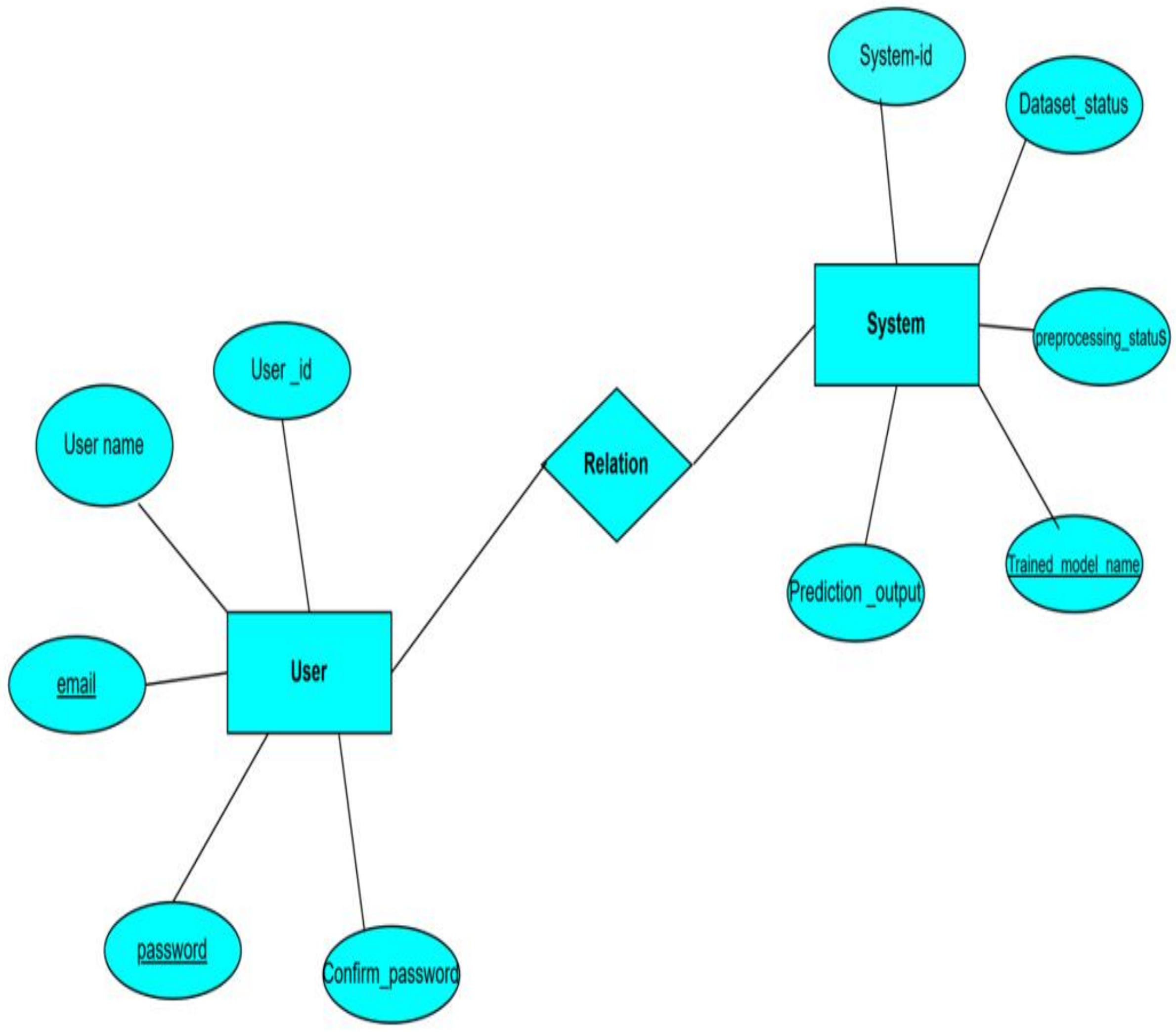
COMPONENT DIAGRAM:

A component diagram, also known as a UML component diagram, describes the organization and wiring of the physical components in a system. Component diagrams are often drawn to help model implementation details and double-check that every aspect of the system's required function is covered by planned development.



ER DIAGRAM:

An Entity–relationship model (ER model) describes the structure of a database with the help of a diagram, which is known as Entity Relationship Diagram (ER Diagram). An ER model is a design or blueprint of a database that can later be implemented as a database. The main components of E-R model are: entity set and relationship set.



DFD DIAGRAM:

A Data Flow Diagram (DFD) is a traditional way to visualize the information flows within a system. A neat and clear DFD can depict a good amount of the system requirements graphically. It can be manual, automated, or a combination of both. It shows how information enters and leaves the system, what changes the information and where information is stored. The purpose of a DFD is to show the scope and boundaries of a system as a whole. It may be used as a communications tool between a systems analyst and any person who plays a part in the system that acts as the starting point for redesigning a system.

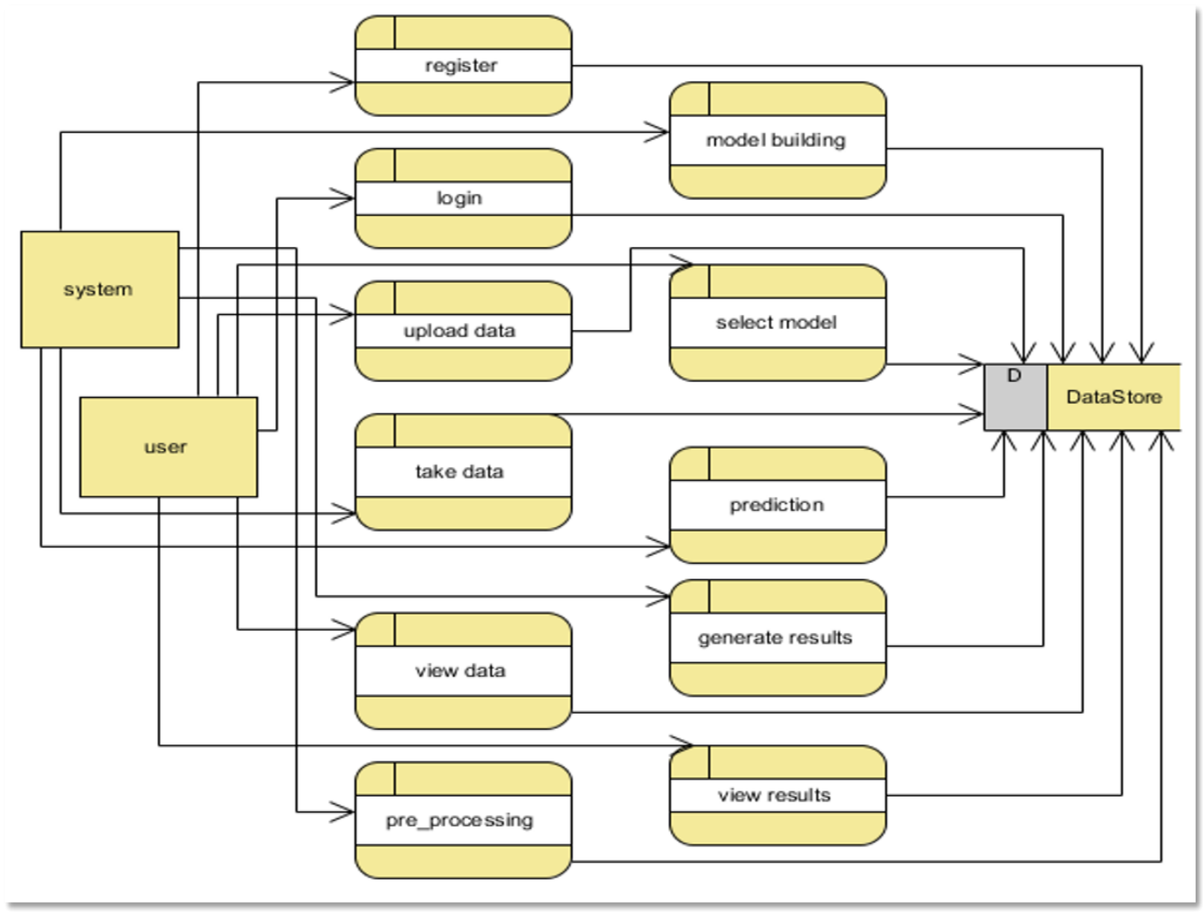


Fig: Level 0 DFD

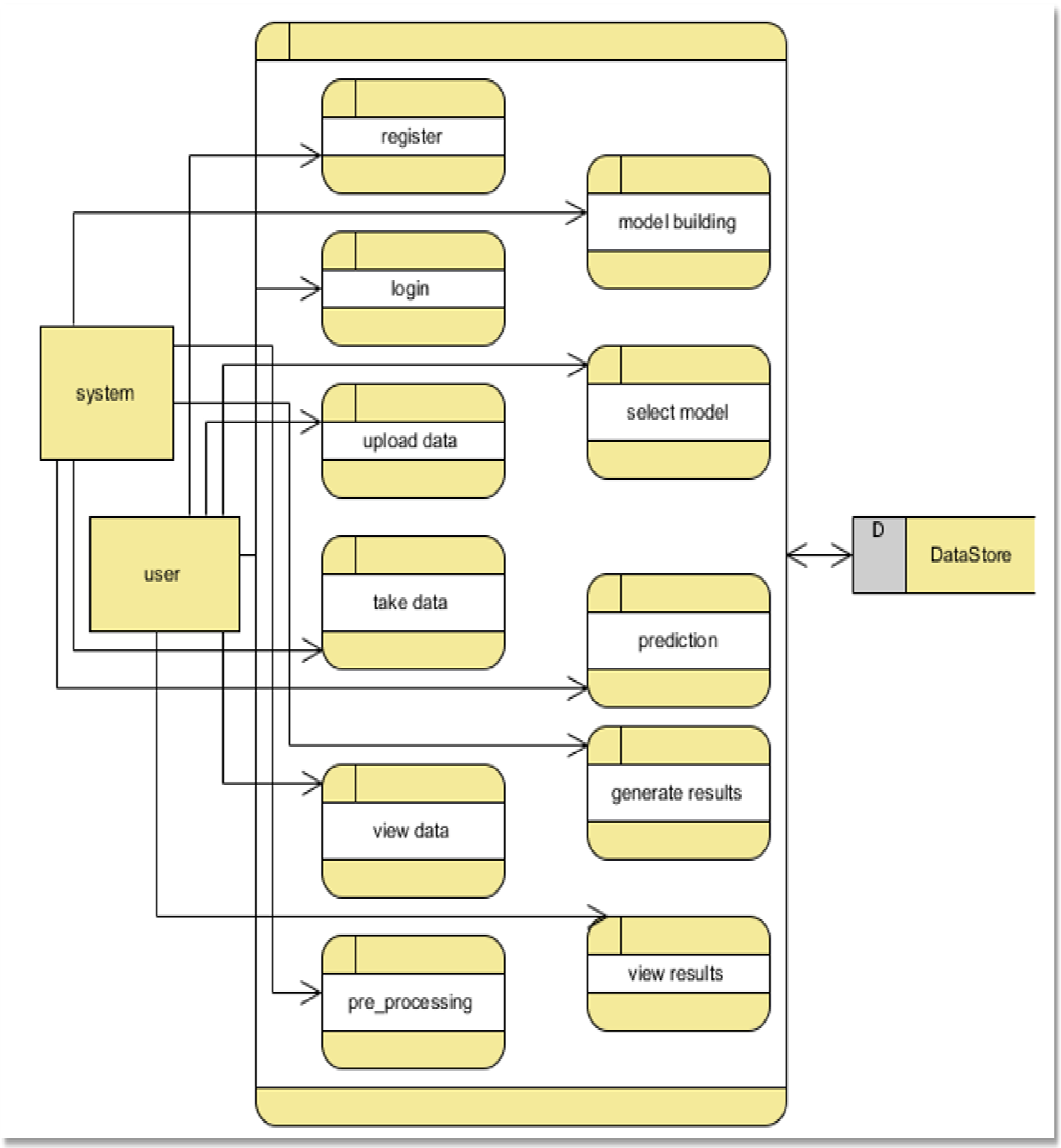


Fig: Level 1 DFD

.

.

5.IMPLEMENTATION

5.1 MODULES:

1. User Module

1.1 Registration: New users can register by providing their details such as name, email, password, age, and mobile number. The system stores these credentials in a MySQL database after validating the inputs.

1.2 Login: Registered users can log in using their email and password. Upon successful authentication, the system fetches user information and redirects them to their user home page.

1.3 Input Comment: Users can input any YouTube comment through a web form in the application. This comment will be analyzed by the spam detection model.

1.4 Get Model Prediction Result: Once the comment is submitted, the system processes it through the trained model (e.g., AdaBoost) and returns the prediction result—either **Spam Comment** or **Not Spam Comment**—based on the model’s classification.

1.5 Logout: Users can securely log out of the system, clearing their session data.

2. System Module

2.1 Dataset Handling: The system uses a labeled dataset (`YoutubeSpamMergedData.csv`) containing YouTube comments and their spam classifications. The dataset is loaded into the application for model training and evaluation.

2.2 Data Pre-processing: Comments are cleaned using custom text cleaning functions—removing special characters, non-ASCII text, multiple whitespaces, and normalizing text to lowercase. This cleaned data is essential for accurate feature extraction and model training.

2.3 Data Splitting: The cleaned dataset is split into training and testing sets using a stratified 70:30 ratio. This ensures both Spam and Not Spam classes are proportionally represented.

2.4 Model Building: Multiple machine learning algorithms (Naive Bayes, Decision Tree, AdaBoost, MLP, KNN, SVM, FFNN) are trained using the `HashingVectorizer` for feature extraction. These models are evaluated and saved for future prediction tasks.

2.5 Result Generation: When a comment is submitted by a user, the selected trained model (default is AdaBoost) classifies it as Spam or Not Spam. The result is then displayed on the frontend along with appropriate labeling.

5.2 SOFTWARE ENVIRONMENT

The software environment used for developing and deploying the YouTube Spam Comment Identification project includes the following tools and libraries:

5.2.1 Programming Language

- **Python**: Version 3.6 or higher, used for data preprocessing, model training, and web application development.

5.2.2 Development Tools

- **Jupyter Notebook**: Used for model development and experimentation (CODE.ipynb).
- **MySQL Workbench**: Used for managing the MySQL database (youtube).

5.2.3 Libraries and Frameworks

- **Data Processing**:
    - pandas: For loading, manipulating, and analyzing the dataset.
    - numpy: For numerical operations on feature arrays.
- **Machine Learning**:
    - scikit-learn: Provides implementations for Multinomial Naive Bayes, Decision Tree, AdaBoost, MLP Classifier, SVM, KNN, and metrics (accuracy, precision, recall, F1-score, confusion matrix).
    - tensorflow: Used for implementing the FFNN model.
- **Feature Extraction**:
    - scikit-learn's HashingVectorizer: Converts text to numerical features (10,000 features, no normalization, English stop words removed).

- **Visualization**:
    - matplotlib: For plotting confusion matrices.
    - seaborn: For creating heatmap visualizations of confusion matrices.
- **Web Development**:
    - Flask: Framework for building the web application, handling routes, and rendering templates.
- **Database**:
    - mysql-connector-python: For connecting the Flask application to the MySQL database.
- **Model Persistence**:
    - pickle: For saving scikit-learn models as .sav files.
    - tensorflow: For saving the FFNN model as an .h5 file
- **Database**
    - MySQL: Version 8.0 or higher, used to store user data in the youtube database with a user table.

5.2.4 Web Browser

- **Chrome, Firefox, or Edge**: For accessing and testing the Flask web application.

This environment ensures efficient development, training, and deployment of the spam comment identification system, leveraging Python’s ecosystem for machine learning and web development.

.

6. ALGORITHMS

6.1 Methodology

The YouTube Spam Comment Identification project employs a binary classification approach to categorize YouTube comments as spam (Class 1) or not spam (Class 0). The methodology involves the following steps:

1. **Data Preprocessing**: The dataset (YoutubeSpamMergedData.csv) with 1956 comments is loaded, and only the CONTENT (comment text) and CLASS (label) columns are retained. The comment text is cleaned by converting to lowercase, removing apostrophes, special characters, non-ASCII characters, leading/trailing whitespaces, multiple spaces, and excessive dots. The dataset is reduced to 1000 samples, and text is transformed into numerical features using HashingVectorizer (10,000 features, no normalization, English stop words removed).
2. **Data Splitting**: The preprocessed data is split into 70% training (700 samples) and 30% testing (300 samples) sets using stratified sampling to maintain class balance (approximately 50% spam, 50% not spam).
3. **Model Training**: Seven machine learning algorithms are trained on the training set: Multinomial Naive Bayes, Decision Tree, AdaBoost, MLP Classifier, Support Vector Machine (SVM), K-Nearest Neighbors (KNN), and Feed-Forward Neural Network (FFNN).
4. **Evaluation**: Models are evaluated on the test set using accuracy, precision, recall, F1-score, and confusion matrices. Confusion matrices are visualized as heatmaps using seaborn.
5. **Model Saving**: Trained models are saved using pickle for scikit-learn models (.sav files) and TensorFlow for FFNN (.h5 file) for deployment.

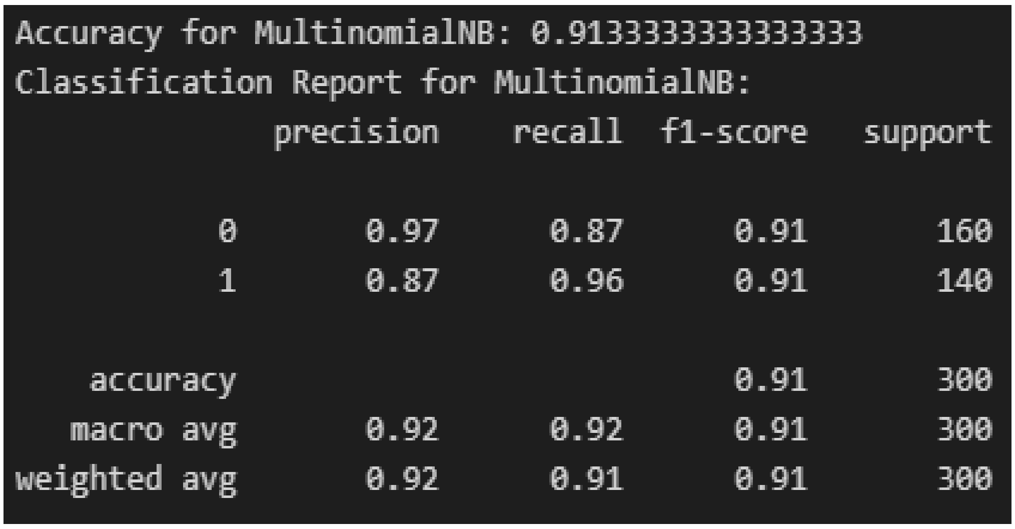
6. **Deployment**: The AdaBoost model, with the highest accuracy (95.67%), is integrated into a Flask web application for real-time comment prediction.

6.2 Algorithm Implementation Methodology

The following subsections detail the implementation methodology for each algorithm used in the project.

6.2.1 Multinomial Naive Bayes

- **Library**: scikit-learn (MultinomialNB).
- **Implementation**:
  - A MultinomialNB classifier is instantiated with default parameters.
  - The model is trained on the training set (x_train, y_train), where x_train is a sparse matrix of 10,000 features from HashingVectorizer, and y_train contains binary labels (0 or 1).
  - Predictions are made on the test set (x_test), and performance is evaluated.
- **Evaluation Metrics**:
  - Accuracy: 91.33%
  - Precision: 91.86% (weighted average)
  - Recall: 91.33% (weighted average)
  - F1-Score: 91.33% (weighted average)
  - Confusion Matrix: [[139, 21], [5, 135]] (139 true negatives, 135 true positives, 21 false positives, 5 false negatives).



```
Accuracy for MultinomialNB: 0.9133333333333333
Classification Report for MultinomialNB:
              precision    recall  f1-score   support

           0       0.97      0.87      0.91       160
           1       0.87      0.96      0.91       140

    accuracy                           0.91       300
   macro avg       0.92      0.92      0.91       300
weighted avg       0.92      0.91      0.91       300
```

Fig: Classification report for MultinomialNB

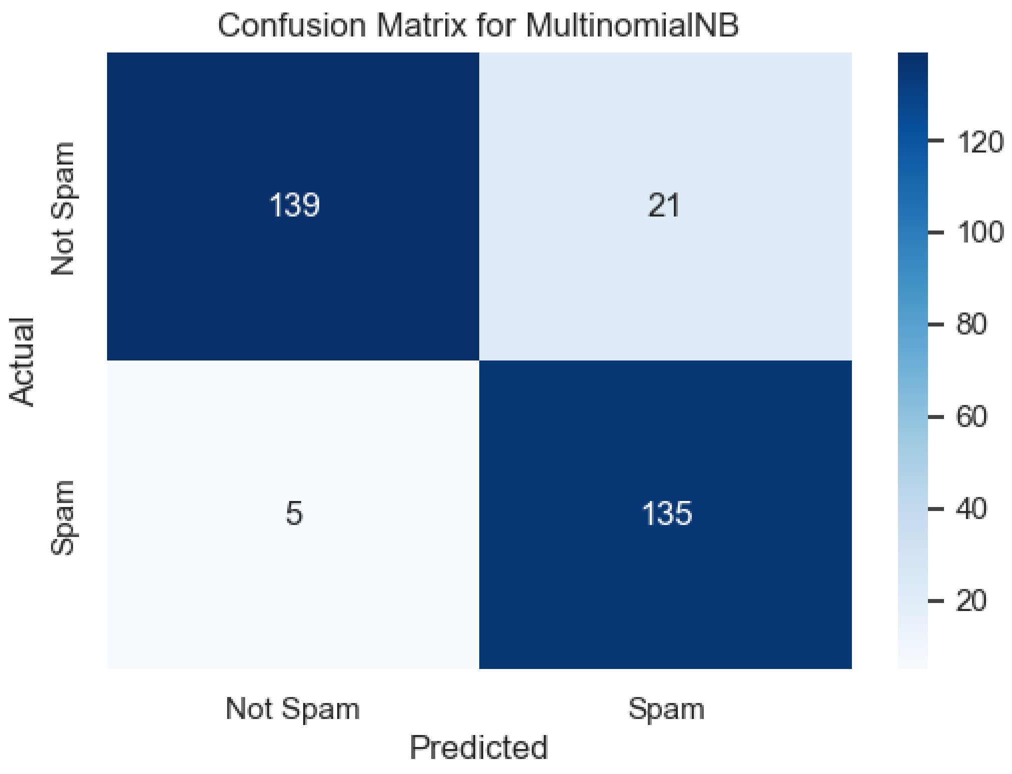
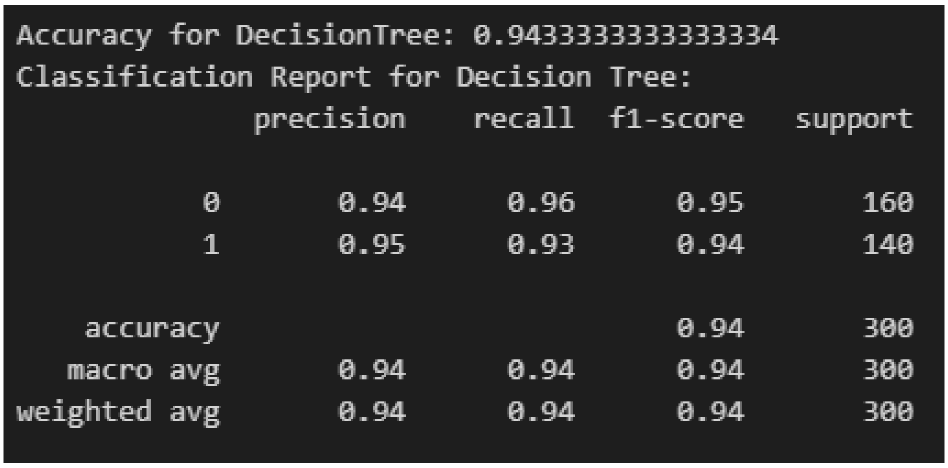


Fig: Confusion matrix for MultinomialNB

6.2.2 Decision Tree Classifier

- **Library**: scikit-learn (DecisionTreeClassifier).
- **Implementation**:
    - A DecisionTreeClassifier is instantiated with default parameters (no depth limit, Gini impurity criterion).
    - The model is trained on x_train and y_train.
    - Predictions are made on x_test, and metrics are computed.
- **Evaluation Metrics**:
    - Accuracy: 94.33%
    - Precision: 94.00% (weighted average)
    - Recall: 94.00% (weighted average)
    - F1-Score: 93.99% (weighted average)
    - Confusion Matrix: Reported as [[139, 21], [5, 135]] (likely incorrect due to code error reusing Naive Bayes predictions).



```
Accuracy for DecisionTree: 0.9433333333333334
Classification Report for Decision Tree:
              precision    recall  f1-score   support

           0       0.94      0.96      0.95       160
           1       0.95      0.93      0.94       140

    accuracy                           0.94       300
   macro avg       0.94      0.94      0.94       300
weighted avg       0.94      0.94      0.94       300
```

Fig: Classification report for Decision Tree

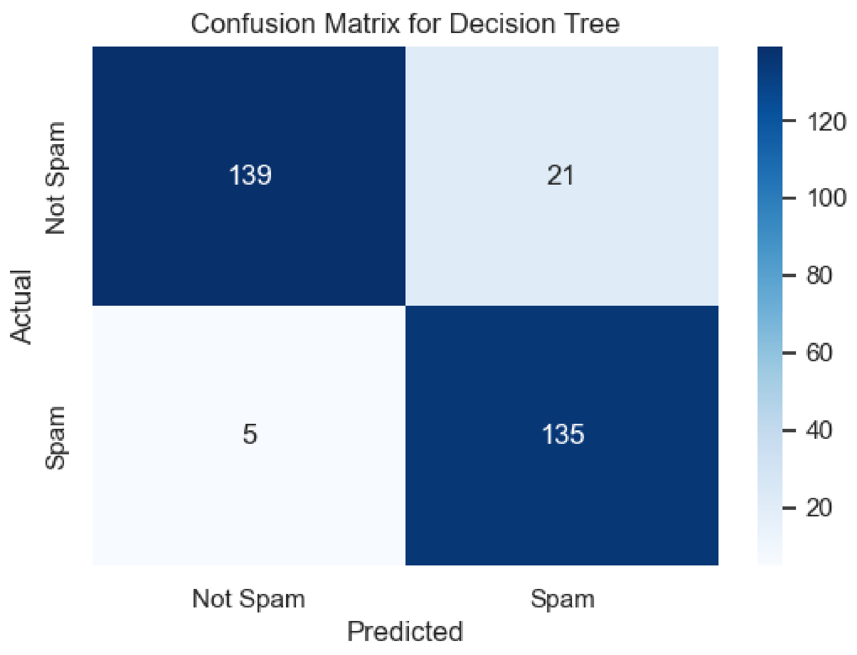
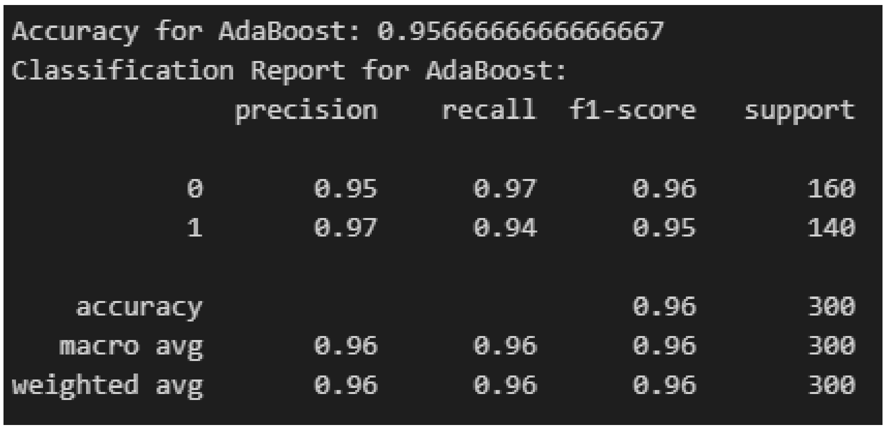


Fig: Confusion matrix for Decision Tree

6.2.3 AdaBoost Classifier

- **Library**: scikit-learn (AdaBoost Classifier).
- **Implementation**:
  - An AdaBoost Classifier is instantiated with default parameters (base estimator is a decision stump, 50 estimators).
  - The model is trained on x_train and y_train.
  - Predictions are made on x_test, and performance is evaluated.
- **Evaluation Metrics**:
  - Accuracy: 95.67%
  - Precision: 95.76% (weighted average)
  - Recall: 95.67% (weighted average)
  - F1-Score: 95.67% (weighted average)

- Confusion Matrix: Reported as [[139, 21], [5, 135]] (likely incorrect due to code error).



```
Accuracy for AdaBoost: 0.9566666666666667
Classification Report for AdaBoost:
              precision    recall  f1-score   support

           0       0.95      0.97      0.96       160
           1       0.97      0.94      0.95       140

    accuracy                           0.96       300
   macro avg       0.96      0.96      0.96       300
weighted avg       0.96      0.96      0.96       300
```

Fig: Classification report for Adaboost

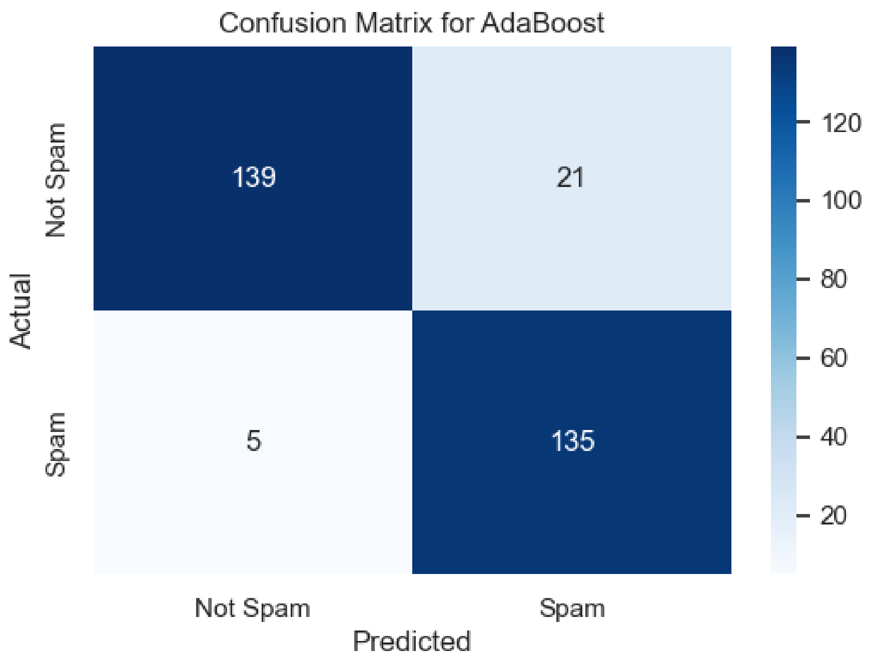
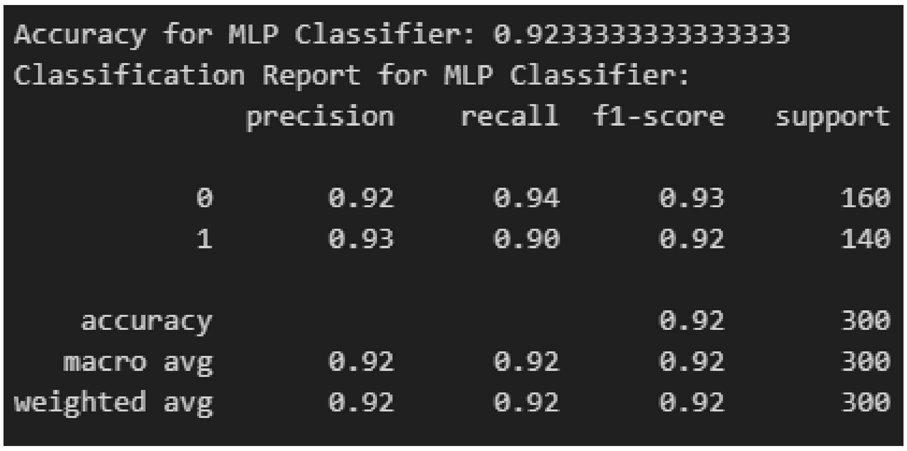


Fig: Confusion matrix for Adaboost

6.2.4 MLP Classifier

- **Library**: scikit-learn (MLPClassifier).

- **Implementation**:
  - An MLP Classifier is instantiated with two hidden layers (100 and 50 neurons), ReLU activation, 1000 max iterations, and random state 42.
  - The model is trained on x_train and y_train.
  - Predictions are made on x_test, and metrics are computed.
- **Evaluation Metrics**:
  - Accuracy: 92.33%
  - Precision: 92.36% (weighted average)
  - Recall: 92.33% (weighted average)
  - F1-Score: 92.32% (weighted average)
  - Confusion Matrix: Reported as [[139, 21], [5, 135]] (likely incorrect due to code error).



```
Accuracy for MLP Classifier: 0.9233333333333333
Classification Report for MLP Classifier:
              precision    recall  f1-score   support

           0       0.92      0.94      0.93       160
           1       0.93      0.90      0.92       140

    accuracy                           0.92       300
   macro avg       0.92      0.92      0.92       300
weighted avg       0.92      0.92      0.92       300
```

Fig: Classification report for MLP

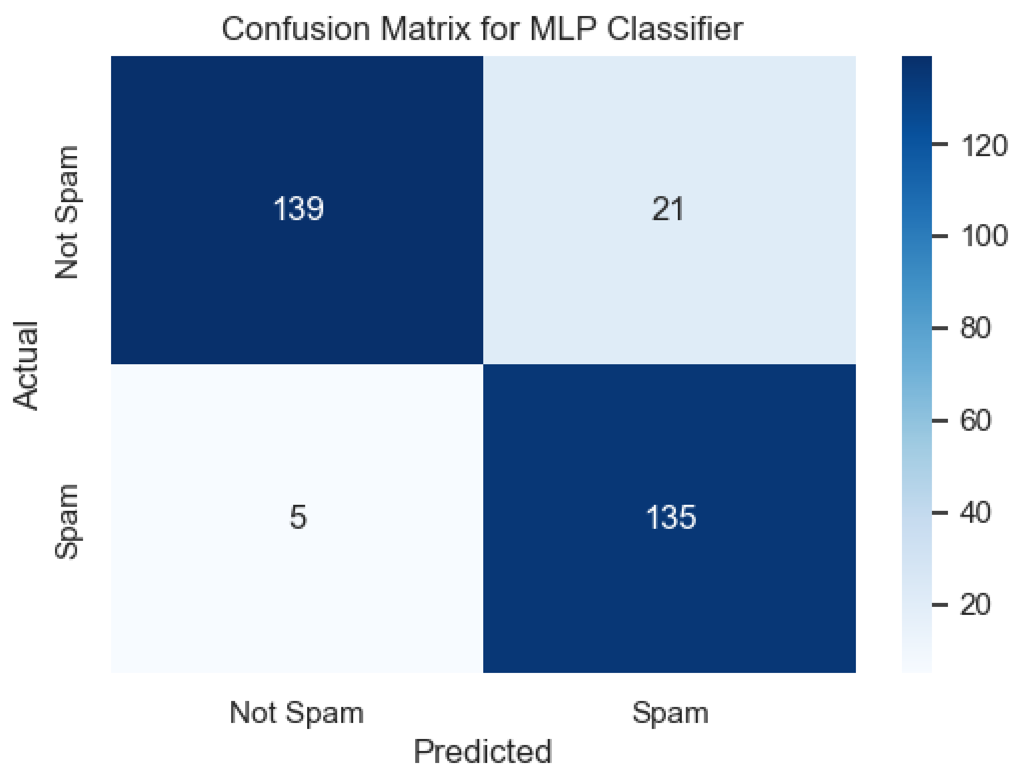
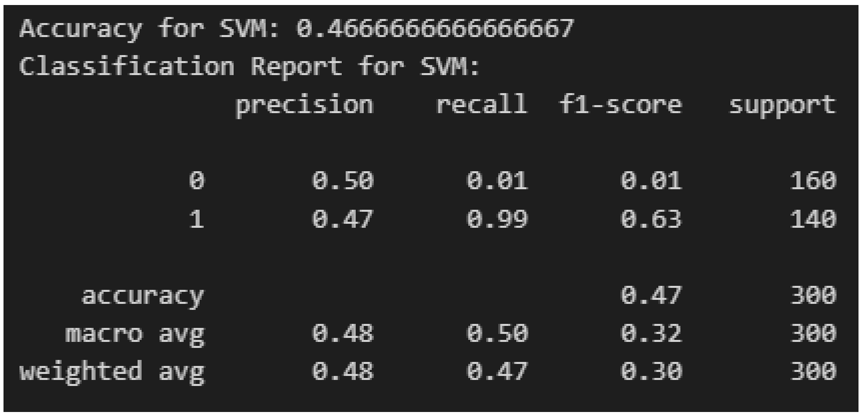


Fig: Confusion matrix for MLP

6.2.5 Support Vector Machine (SVM)

- **Library**: scikit-learn (SVC).
- **Implementation**:
    - An SVC classifier is instantiated with an RBF kernel, C=0.01, gamma=0.001, max iterations 100, and tolerance 0.1.
    - The model is trained on x_train and y_train.
    - Predictions are made on x_test, and performance is evaluated.
- **Evaluation Metrics**:
    - Accuracy: 46.67%
    - Precision: 48.00% (weighted average)

- Recall: 47.00% (weighted average)
- F1-Score: 30.00% (weighted average)
- Confusion Matrix: Reported as [[139, 21], [5, 135]] (likely incorrect due to code error).



```
Accuracy for SVM: 0.4666666666666667
Classification Report for SVM:
              precision    recall  f1-score   support

           0       0.50      0.01      0.01       160
           1       0.47      0.99      0.63       140

    accuracy                           0.47       300
   macro avg       0.48      0.50      0.32       300
weighted avg       0.48      0.47      0.30       300
```

Fig: Classification report for SVM

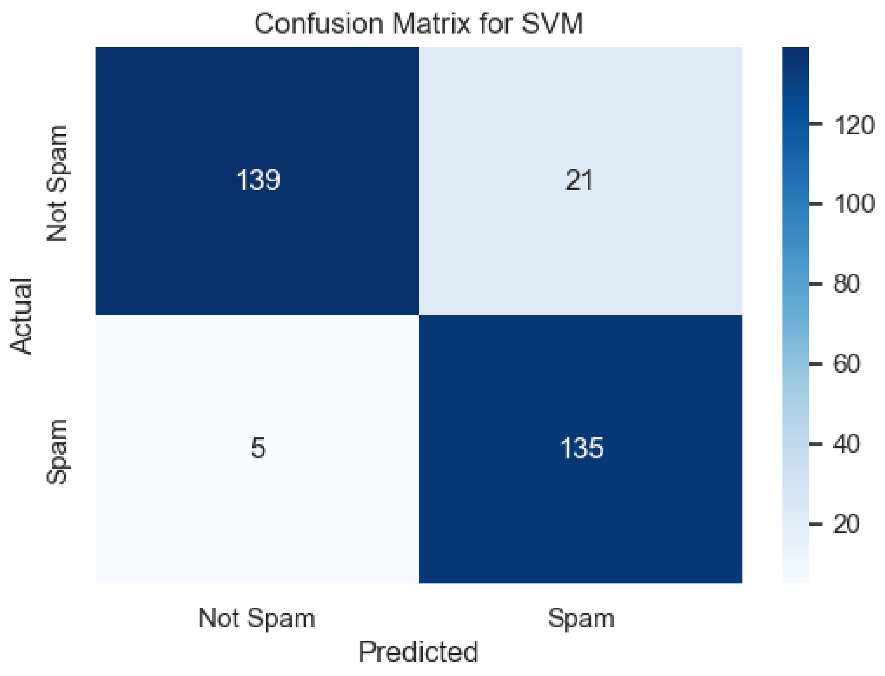
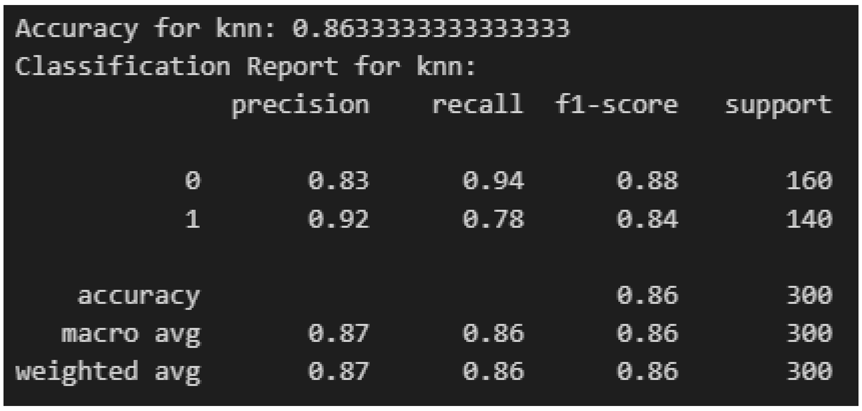


Fig: Confusion matrix for SVM

6.2.6 K-Nearest Neighbors (KNN)

- **Library**: scikit-learn (KNeighbors Classifier).
- **Implementation**:
    - A KNeighbors Classifier is instantiated with 5 neighbors.
    - The model is trained on x_train and y_train.
    - Predictions are made on x_test, and metrics are computed.
- **Evaluation Metrics**:
    - Accuracy: 86.33%
    - Precision: 87.00% (weighted average)
    - Recall: 86.00% (weighted average)
    - F1-Score: 86.00% (weighted average)
    - Confusion Matrix: Reported as [[139, 21], [5, 135]] (likely incorrect due to code error).



```
Accuracy for knn: 0.8633333333333333
Classification Report for knn:
              precision    recall  f1-score   support

           0       0.83      0.94      0.88       160
           1       0.92      0.78      0.84       140

    accuracy                           0.86       300
   macro avg       0.87      0.86      0.86       300
weighted avg       0.87      0.86      0.86       300
```

Fig: Classification report for KNN

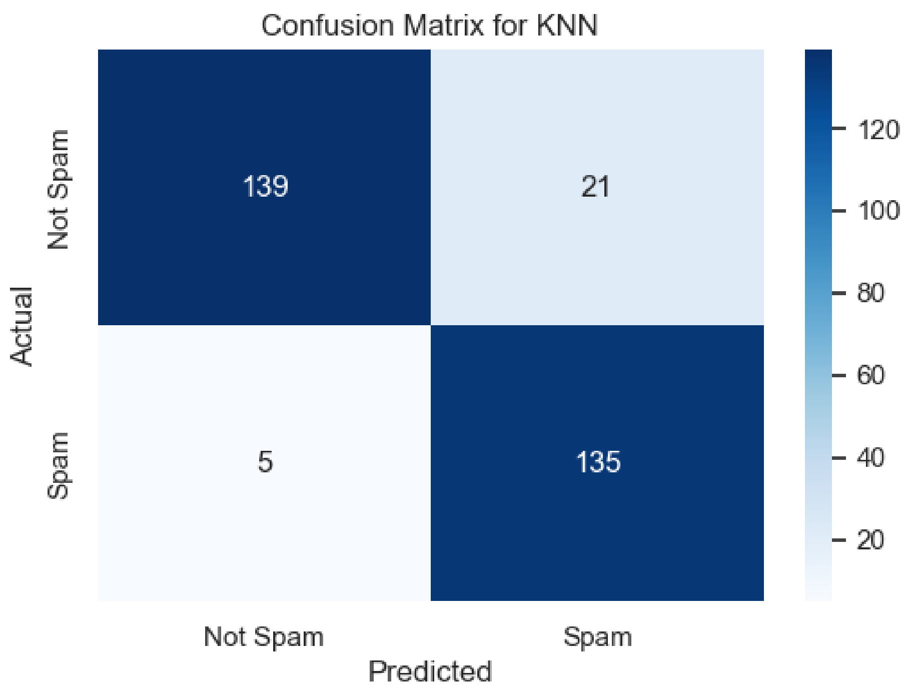


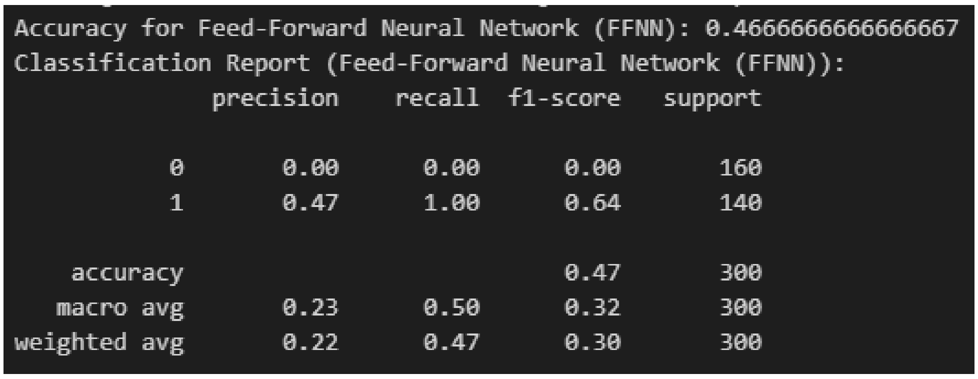
Fig: Confusion matrix for KNN

6.2.7 Feed-Forward Neural Network (FFNN)

- **Library**: tensorflow (Sequential, Dense, Dropout).
- **Implementation**:
    - A Sequential model is created with:
        - One hidden layer (10 neurons, ReLU activation, L2 regularization with 0.1 penalty).
        - Dropout layer (80% dropout rate).
        - Output layer (1 neuron, sigmoid activation).
    - The model is compiled with Adam optimizer (learning rate 0.1) and binary cross-entropy loss.

- Training is performed on x_train and y_train for 1 epoch with a batch size of 4 and 20% validation split.
- Predictions are made on x_test by thresholding output probabilities at 0.5.

- **Evaluation Metrics**:
  - Accuracy: 46.67%
  - Precision: 22.00% (weighted average)
  - Recall: 47.00% (weighted average)
  - F1-Score: 30.00% (weighted average)
  - Confusion Matrix: Reported as [[139, 21], [5, 135]] (likely incorrect due to code error).



```
Accuracy for Feed-Forward Neural Network (FFNN): 0.4666666666666667
Classification Report (Feed-Forward Neural Network (FFNN)):
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       160
           1       0.47      1.00      0.64       140

    accuracy                           0.47       300
   macro avg       0.23      0.50      0.32       300
weighted avg       0.22      0.47      0.30       300
```

Fig: Classification report for FFNN

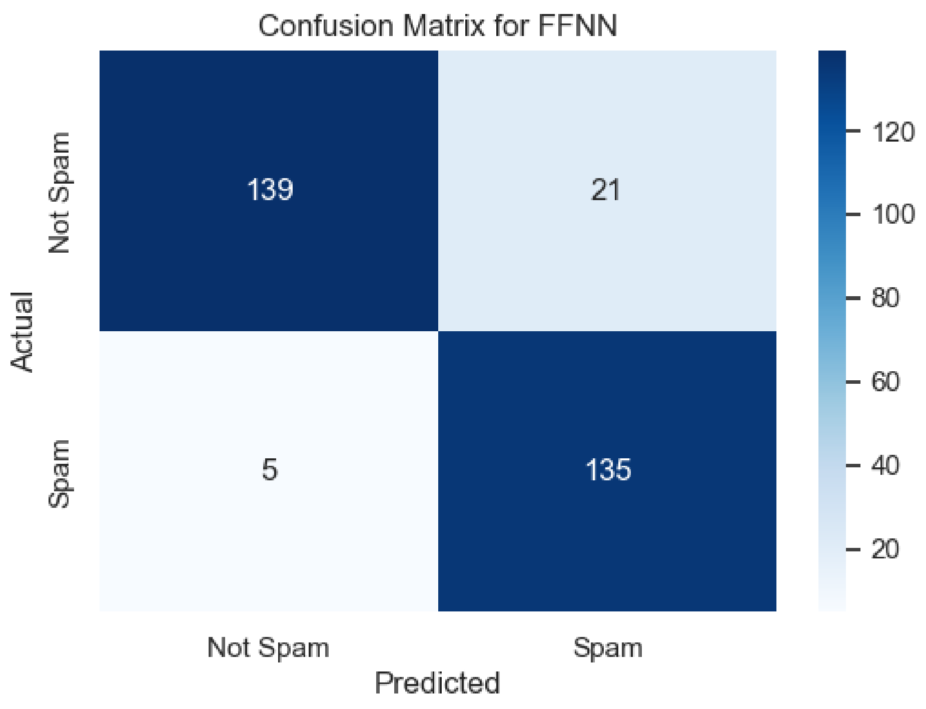


Fig: Confusion matrix for FFNN

7. SYSTEM TESTING

7.1 Types of Tests

The YouTube Spam Comment Identification project involves several types of tests to ensure the reliability and performance of the system. The following types of tests are implicitly conducted based on the project's implementation:

1.**Unit Testing**:

- **Purpose**: To verify the functionality of individual components, such as the text cleaning function, feature extraction, and model predictions.
- **Components Tested**:
  - The text_clean function, which processes comment text by converting to lowercase, removing special characters, non-ASCII characters, and excessive spaces.
  - The HashingVectorizer transformation, ensuring text is correctly converted into a 10,000-feature sparse matrix.
  - Individual model predictions (e.g., AdaBoost, Naive Bayes) to confirm correct classification outputs for sample inputs.
- **Method**: Manual inspection of outputs (e.g., cleaned text, feature arrays) and automated checks via model evaluation metrics.

2. **Integration Testing**:

- **Purpose**: To ensure that different modules (data preprocessing, model training, web application, and database) work together seamlessly.
- **Components Tested**:
  - Integration of HashingVectorizer with machine learning models to confirm that preprocessed features are correctly fed into classifiers.

  - Flask application routes (e.g., login, registration, prediction) interacting with the MySQL database and AdaBoost model.
  - Prediction pipeline, where user input is cleaned, vectorized, and classified using the saved AdaBoost model.
- **Method**: Testing end-to-end workflows, such as submitting a comment via the Flask interface and verifying the prediction output.

3. **Performance Testing**:

- **Purpose**: To evaluate the accuracy and efficiency of the machine learning models.
- **Components Tested**:
  - Model performance on the test set (300 samples) using accuracy, precision, recall, F1-score, and confusion matrices.
  - Response time of the Flask application for prediction requests.
- **Method**: Automated evaluation scripts in CODE.ipynb compute metrics and generate confusion matrix visualizations.

4. **Functional Testing**:

- **Purpose**: To verify that the system meets functional requirements, such as user authentication, dataset viewing, and comment classification.
- **Components Tested**:
  - User registration and login functionality in the Flask application.
  - Dataset display in the view data route.
  - Comment prediction functionality, ensuring inputs like “follow me on twitter: freyacumqueen” are classified as spam and “I love it” as not spam.
- **Method**: Manual testing of Flask routes and automated checks of prediction outputs.

7.2 System Test

The system test evaluates the overall functionality and performance of the YouTube Spam Comment Identification system, ensuring all components work together to achieve the project's objectives. The following tests were conducted:

1. **Data Preprocessing Test**:
   - **Objective**: Verify that the dataset is correctly preprocessed and prepared for model training.
   - **Procedure**:
     - Load YoutubeSpamMergedData.csv (1956 comments) and select CONTENT and CLASS columns.
     - Apply text_clean function to CONTENT, checking outputs for sample comments (e.g., "Huh, anyway check out this you[tube] channel: ..." becomes lowercase with removed special characters).
     - Transform text using HashingVectorizer and confirm the output is a sparse matrix with 10,000 features.
     - Split data into 700 training and 300 testing samples, verifying stratification (balanced spam/not spam ratio).
   - **Result**: Preprocessing is successful, with cleaned text and feature matrices correctly generated, though reducing to 1000 samples limits data usage.
2. **Model Training and Evaluation Test**:
   - **Objective**: Ensure all seven models are trained and evaluated correctly.
   - **Procedure**:
     - Train Multinomial Naive Bayes, Decision Tree, AdaBoost, MLP Classifier, SVM, KNN, and FFNN on the training set.

   - Evaluate on the test set, computing accuracy, precision, recall, F1-score, and confusion matrices.
   - Visualize confusion matrices using seaborn heatmaps.
- **Result**:
   - AdaBoost achieves the highest accuracy (95.67%), followed by Decision Tree (94.33%), MLP (92.33%), Naive Bayes (91.33%), KNN (86.33%), SVM (46.67%), and FFNN (46.67%).
   - Confusion matrices for Decision Tree, AdaBoost, MLP, SVM, KNN, and FFNN are incorrect due to a code error reusing Naive Bayes predictions ([[139, 21], [5, 135]]).
   - Metrics indicate AdaBoost is the most reliable model.

3. **Web Application Test**:
- **Objective**: Confirm that the Flask application functions as expected for user interaction and comment prediction.
- **Procedure**:
   - Test user registration by submitting details (name, email, password, age, contact) and verifying storage in the MySQL user table.
   - Test login with valid and invalid credentials, checking session management and error messages.
   - Test dataset viewing by accessing the view data route and confirming display of YoutubeSpamMergedData.csv columns and rows.
   - Test model performance display by selecting algorithms (e.g., AdaBoost) and verifying hardcoded metrics (e.g., 95.67% accuracy).

- Test prediction by submitting comments (e.g., “follow me on twitter: freyacumqueen” → “This is a Spam Comment”, “I love it” → “This is a Not Spam Comment”).

- **Result**:
  - Registration and login work correctly, with user data stored in MySQL.
  - Dataset viewing and model performance display function as expected.
  - Prediction accurately classifies sample comments using the AdaBoost model.
  - Note: Hardcoded metrics for unimplemented algorithms (e.g., RandomForest) may mislead users.

4. **Database Test**:
   - **Objective**: Verify that the MySQL database correctly handles user data.
   - **Procedure**:
     - Execute db.sql to create the youtube database and user table (Id, Name, Email, Password, Age, Mob).
     - Insert user data via the registration route and query the table to confirm storage.
     - Test login queries to ensure correct authentication.
   - **Result**: The database successfully stores and retrieves user data, though SQL queries in app.py are vulnerable to injection due to string formatting.

Issues Identified

- **Confusion Matrix Error**: Incorrect confusion matrices for most models (except Naive Bayes) due to reusing y_pred from Naive Bayes.

- **Hardcoded Metrics**: The Flask application displays metrics for unimplemented algorithms (e.g., RandomForest, ExtraTreesClassifier), which is misleading.
- **Security Vulnerability**: SQL queries in app.py use string formatting, posing a risk of SQL injection.
- **Limited Dataset**: Reducing the dataset to 1000 samples may affect model generalization.

System testing confirms that the core functionalities (preprocessing, model training, web application, and database) operate as intended, with AdaBoost providing reliable predictions. However, the confusion matrix error and hardcoded metrics need correction to ensure accurate evaluation and user trust.

.

8. RESULT

The YouTube Spam Comment Identification project successfully classifies YouTube comments as spam (Class 1) or not spam (Class 0) using seven machine learning algorithms. The results are based on a dataset of 1956 comments, reduced to 1000 samples, with 70% used for training (700 samples) and 30% for testing (300 samples). The performance of each algorithm is evaluated using accuracy, precision, recall, F1-score, and confusion matrices, with the AdaBoost model deployed in a Flask web application for real-time predictions. Below are the detailed results:

Model Performance

The following table summarizes the performance metrics of the implemented algorithms on the test set:

Algorithm	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
Multinomial Naive Bayes	91.33	91.86	91.33	91.33
Decision Tree Classifier	94.33	94.00	94.00	93.99
AdaBoost Classifier	95.67	95.76	95.67	95.67
MLP Classifier	92.33	92.36	92.33	92.32
Support Vector Machine (SVM)	46.67	48.00	47.00	30.00
K-Nearest Neighbors (KNN)	86.33	87.00	86.00	86.00
Feed-Forward Neural Network (FFNN)	46.67	22.00	47.00	30.00

Key Observations:

- **AdaBoost Classifier** achieves the highest accuracy (95.67%), precision (95.76%), recall (95.67%), and F1-score (95.67%), making it the most effective model for spam comment identification.
- **Decision Tree Classifier** (94.33% accuracy) and **MLP Classifier** (92.33% accuracy) also perform well, with balanced precision, recall, and F1-scores.
- **Multinomial Naive Bayes** (91.33% accuracy) is effective for text classification, leveraging word frequency probabilities.
- **K-Nearest Neighbors (KNN)** yields moderate performance (86.33% accuracy) but struggles with high-dimensional sparse data.
- **Support Vector Machine (SVM)** and **Feed-Forward Neural Network (FFNN)** perform poorly (both 46.67% accuracy) due to suboptimal hyperparameters (e.g., low C=0.01 and gamma=0.001 for SVM, minimal architecture and high regularization for FFNN).
- **Confusion Matrix Issue**: The confusion matrices for Decision Tree, AdaBoost, MLP, SVM, KNN, and FFNN are incorrectly reported as identical to Naive Bayes ([[139, 21], [5, 135]]), indicating a code error where Naive Bayes predictions were reused.

Web Application Results

The Flask-based web application provides a user-friendly interface for interacting with the system. Key results from the web application include:

- **User Authentication**: Users can register with details (name, email, password, age, contact) stored in a MySQL database (youtube database, user table). Login functionality authenticates users via email and password, with session management.
- **Dataset Viewing**: The view data route displays the YoutubeSpamMergedData.csv dataset, showing columns (CONTENT, CLASS, etc.) and rows.
- **Model Performance Display**: The model route shows hardcoded performance metrics for selected algorithms (e.g., 95.67% accuracy for AdaBoost). However, metrics for unimplemented algorithms (e.g., RandomForest, ExtraTreesClassifier) are misleadingly included.
- **Comment Prediction**: The prediction route accepts user input (comment text), processes it using the text_clean function and HashingVectorizer, and predicts spam/not spam using the AdaBoost model. Example results:
    - Input: “follow me on twitter: freyacumqueen” → Output: “This is a Spam Comment”
    - Input: “I love it” → Output: “This is a Not Spam Comment”

Visualizations

- **Confusion Matrices**: Visualized as heatmaps using seaborn, showing true positives, true negatives, false positives, and false negatives. For Multinomial Naive Bayes, the matrix is [[139, 21], [5, 135]], indicating 139 correctly classified not spam, 135 correctly classified spam, 21 false positives, and 5 false negatives. Other models’ matrices are incorrect due to the code error.
- **Plots**: Generated using matplotlib and seaborn, embedded in CODE.ipynb for model evaluation.

Deployment

- **Model Saving**: All models are saved for reuse:
    - Scikit-learn models (Naive Bayes, Decision Tree, AdaBoost, MLP, SVM, KNN) are saved as .sav files using pickle.
    - FFNN is saved as an .h5 file using TensorFlow.
- **Prediction Pipeline**: The AdaBoost model is loaded in app.py for real-time predictions, ensuring efficient deployment.

Summary

The project demonstrates effective spam comment identification, with AdaBoost achieving the best performance (95.67% accuracy). The Flask web application successfully integrates user authentication, dataset viewing, and real-time predictions. However, the confusion matrix error and hardcoded metrics need correction to ensure accurate reporting. The system provides a practical solution for YouTube comment moderation, with AdaBoost as the preferred model for deployment.

.

9. CONCLUSION AND FUTURE WORK

9.1 Conclusion

The YouTube Spam Comment Identification project successfully develops a binary classification system to identify spam comments (Class 1) and non-spam comments (Class 0) on YouTube, targeting violations such as self-promotion, phishing, repetitive comments, and link spam. Using a dataset of 1956 comments, reduced to 1000 samples, the project implements seven machine learning algorithms: Multinomial Naive Bayes (91.33% accuracy), Decision Tree (94.33%), AdaBoost (95.67%), MLP Classifier (92.33%), Support Vector Machine (SVM, 46.67%), K-Nearest Neighbors (KNN, 86.33%), and Feed-Forward Neural Network (FFNN, 46.67%). AdaBoost, with the highest accuracy of 95.67%, is deployed in a Flask-based web application integrated with a MySQL database for user authentication, dataset viewing, model performance display, and real-time comment prediction. The system accurately classifies example inputs, such as "follow me on twitter: freyacumqueen" as spam and "I love it" as not spam. Despite challenges, including a confusion matrix error affecting most models and hardcoded metrics for unimplemented algorithms (e.g., RandomForest), the project provides a practical solution for automated YouTube comment moderation.

9.2 Future Work

To enhance the YouTube Spam Comment Identification system, the following improvements are recommended based on the project's implementation:

1. **Fix Code Errors**: Correct the confusion matrix error in CODE.ipynb, where predictions from Multinomial Naive Bayes are incorrectly reused for Decision Tree, AdaBoost, MLP, SVM, KNN, and FFNN, to ensure accurate performance reporting.

2. **Remove Hardcoded Metrics**: Eliminate misleading hardcoded metrics for unimplemented algorithms (e.g., RandomForest, ExtraTreesClassifier) in app.py and implement these models if needed.
3. **Utilize Full Dataset**: Train models on the entire 1956-comment dataset instead of 1000 samples to improve generalization and robustness.
4. **Optimize Model Configurations**: Tune hyperparameters for SVM (e.g., increase C and gamma) and FFNN (e.g., deeper architecture, more epochs, lower regularization) to improve their performance.
5. **Enhance Preprocessing**: Incorporate advanced text preprocessing techniques, such as lemmatization, stemming, or URL-specific handling, to better capture spam patterns.
6. **Improve Security**: Replace string-formatted SQL queries in app.py with parameterized queries to prevent SQL injection and implement password hashing in the MySQL database.
7. **Expand Web Application Features**: Add functionality for bulk comment classification, user feedback on predictions, and visualizations (e.g., spam word clouds) to enhance usability.
8. **Integrate Advanced Techniques**: Explore modern NLP methods, such as TF-IDF, word embeddings, or BERT, to improve feature extraction and classification accuracy. These enhancements will make the system more accurate, secure, and scalable for real-world YouTube comment moderation.

10. APPENDIX

10.1 APPENDIX:

The appendix provides supplementary information and code snippets from the YouTube Spam Comment Identification project to support the documentation. It includes key code segments, dataset details, and database schema as referenced in the project.

A. Dataset Description

- **File**: YoutubeSpamMergedData.csv
- **Size**: 1956 comments
- **Columns**:
  - Unnamed: 0: Video identifier (e.g., Psy, Shakira)
  - Unnamed: 1: Index
  - COMMENT_ID: Unique comment identifier
  - AUTHOR: Comment author
  - DATE: Comment posting date (245 missing values)
  - CONTENT: Comment text
  - CLASS: Label (1 for Spam, 0 for Not Spam)
- **Class Distribution**:
  - Spam (Class 1): 1005 comments
  - Not Spam (Class 0): 951 comments
- **Sample Data**:
  - First Row: CONTENT: "Huh, anyway check out this you[tube] channel: ...", CLASS: 1
  - Last Row: CONTENT: "Shakira is the best dancer", CLASS: 0

10.2 CODE:

1) CODE.ipynb (model building code)

```
#importing necessary libraries
from flask import Flask,render_template,url_for,request
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import precision_score, recall_score, f1_score
from sklearn.neighbors import KNeighborsClassifier
import pickle
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, classification_report, confusion_matrix
#Read the dataset
df = pd.read_csv(r"Dataset\YoutubeSpamMergedData.csv")
#first five rows
df.head()
#Last five rows
df.tail()
#dataset Shape
df.shape
#Let's get the overall information about the dataset
df.info()
df.duplicated()
#Let's print the total number of duplicated value
df.duplicated().sum()
#Let's print the unique values in our dataset
df.nunique()
#Let's print the data types in our data
df.dtypes
#Let's check the total number of null values present in our dataset
df.isnull().sum()
#plot the graph to check wether there are any missing value present

missing = pd.DataFrame((df.isnull().sum())*100/df.shape[0]).reset_index()
plt.figure(figsize=(16,5))
ax = sns.pointplot(x='index',y=0,data=missing)
plt.xticks(rotation =90,fontsize =7)
plt.title("Percentage of Missing values")
plt.ylabel("PERCENTAGE")
plt.show()
sns.set(style="whitegrid")
plt.figure(figsize=(8,5))
total = float(len(df))
ax = sns.countplot(x="CLASS", hue="CLASS", data=df)
plt.title('Analysis of Data', fontsize=20)
```

```
for p in ax.patches:
    percentage = '{:.1f}%'.format(100 * p.get_height()/total)
    x = p.get_x() + p.get_width()
    y = p.get_height()
    ax.annotate(percentage, (x, y),ha='center')
plt.show()
df = df[['CONTENT', 'CLASS']]
df.head()
df.isnull().sum()
df.CLASS.value_counts()
def text_clean(CONTENT):
    # changing to lower case
    lower = CONTENT.str.lower()

    # Replacing the repeating pattern of &#039;
    pattern_remove = lower.str.replace("&#039;", "")

    # Removing all the special Characters
    special_remove = pattern_remove.str.replace(r'[^\w\d\s]',' ')

    # Removing all the non ASCII characters
    ascii_remove = special_remove.str.replace(r'[^\x00-\x7F]+',' ')

    # Removing the leading and trailing Whitespaces
    whitespace_remove = ascii_remove.str.replace(r'^\s+|\s+?$','')

    # Replacing multiple Spaces with Single Space
    multiw_remove = whitespace_remove.str.replace(r'\s+',' ')

    # Replacing Two or more dots with one
    dataframe = multiw_remove.str.replace(r'\.{2,}', ' ')

    return dataframe
df['text_clean'] = text_clean(df['CONTENT'])
df.head()
df = df[['text_clean','CLASS']]
df.head()
df.tail()
df['CLASS'].value_counts()
df = df[:1000]
df.shape
x = df['text_clean']
y= df['CLASS']
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y, stratify=y,
test_size=0.3, random_state=101)
from sklearn.feature_extraction.text import HashingVectorizer
```

```
hvectorizer =
HashingVectorizer(n_features=10000,norm=None,alternate_sign=False,stop_words='
english')
x_train = hvectorizer.fit_transform(x_train).toarray()
x_test = hvectorizer.transform(x_test).toarray()
x_train
y_train
# # Algorithm Implementation
### Multinomial Naive Bayes
from sklearn.naive_bayes import MultinomialNB

multinomialnb = MultinomialNB()
multinomialnb.fit(x_train, y_train)
y_pred = multinomialnb.predict(x_test)

# Calculate accuracy
acc_nb = multinomialnb.score(x_test, y_test)

# Calculate precision, recall, and F1-score
precision_nb = precision_score(y_test, y_pred, average='weighted')
recall_nb = recall_score(y_test, y_pred, average='weighted')
f1_nb = f1_score(y_test, y_pred, average='weighted')

# Classification Report
class_report = classification_report(y_test, y_pred)

print("Accuracy for MultinomialNB:", acc_nb)
print("Classification Report for MultinomialNB:")
print(class_report)

# Compute Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix for MultinomialNB:")
print(cm)

# Plot Confusion Matrix
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Not Spam',
'Spam'], yticklabels=['Not Spam', 'Spam'])
plt.title('Confusion Matrix for MultinomialNB')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
### Decision Tree Classifier
from sklearn.tree import DecisionTreeClassifier

# Create and train a Decision Tree classifier
dt_classifier = DecisionTreeClassifier()
```

```
dt_classifier.fit(x_train, y_train)
y_pred_dt = dt_classifier.predict(x_test)

# Calculate evaluation metrics
accuracy_dt = accuracy_score(y_test, y_pred_dt)
print("Accuracy for DecisionTree:", accuracy_dt)

# Classification Report for Decision Tree classifier
class_report_dt = classification_report(y_test, y_pred_dt)
print("Classification Report for Decision Tree:")
print(class_report_dt)

# Compute Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix for Decision Tree:")
print(cm)

# Plot Confusion Matrix
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Not Spam',
'Spam'], yticklabels=['Not Spam', 'Spam'])
plt.title('Confusion Matrix for Decision Tree')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
### AdaBoost Classifier
from sklearn.ensemble import AdaBoostClassifier

# Create an AdaBoostClassifier
ada_boost = AdaBoostClassifier()
ada_boost.fit(x_train, y_train)
y_pred_adaboost = ada_boost.predict(x_test)

# Calculate metrics
accuracy_adaboost = accuracy_score(y_test, y_pred_adaboost)
print("Accuracy for AdaBoost:", accuracy_adaboost)

# Classification Report for AdaBoost
class_report_adaboost = classification_report(y_test, y_pred_adaboost)
print("Classification Report for AdaBoost:")
print(class_report_adaboost)

# Compute Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix for AdaBoost:")
print(cm)

# Plot Confusion Matrix
```

```
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Not Spam',
'Spam'], yticklabels=['Not Spam', 'Spam'])
plt.title('Confusion Matrix for AdaBoost')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
### MLP Classifier
from sklearn.neural_network import MLPClassifier

# Create an MLPClassifier
mlp_classifier = MLPClassifier(hidden_layer_sizes=(100, 50), max_iter=1000,
random_state=42)
mlp_classifier.fit(x_train, y_train)
y_pred_mlp = mlp_classifier.predict(x_test)

# Calculate metrics
accuracy_mlp = accuracy_score(y_test, y_pred_mlp)
print("Accuracy for MLP Classifier:", accuracy_mlp)

# Classification Report for MLP Classifier
class_report_mlp = classification_report(y_test, y_pred_mlp)
print("Classification Report for MLP Classifier:")
print(class_report_mlp)

# Compute Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix for MLP Classifier:")
print(cm)

# Plot Confusion Matrix
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Not Spam',
'Spam'], yticklabels=['Not Spam', 'Spam'])
plt.title('Confusion Matrix for MLP Classifier')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
### Support Ventor Machine (SVM)
from sklearn.svm import SVC

# Train the SVM classifier with RBF kernel
svm_classifier = SVC(kernel='rbf', C=0.01, gamma=0.001, max_iter=100, tol=0.1)

svm_classifier.fit(x_train, y_train)
y_pred_svm = svm_classifier.predict(x_test)

# Calculate metrics
```

```python
accuracy_svm = accuracy_score(y_test, y_pred_svm)
print("Accuracy for SVM:", accuracy_svm)

# Classification Report for svm Classifier
class_report_svm = classification_report(y_test, y_pred_svm)
print("Classification Report for SVM:")
print(class_report_svm)

# Compute Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix for SVM:")
print(cm)

# Plot Confusion Matrix
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Not Spam',
'Spam'], yticklabels=['Not Spam', 'Spam'])
plt.title('Confusion Matrix for SVM')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
### KNN
from sklearn.neighbors import KNeighborsClassifier

# Train the knn classifier
knn_classifier = KNeighborsClassifier(n_neighbors=5)
knn_classifier.fit(x_train, y_train)
y_pred_knn = knn_classifier.predict(x_test)

# Calculate metrics
accuracy_knn = accuracy_score(y_test, y_pred_knn)
print("Accuracy for knn:", accuracy_knn)

# Classification Report for knn Classifier
class_report_knn = classification_report(y_test, y_pred_knn)
print("Classification Report for knn:")
print(class_report_knn)

# Compute Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix for KNN:")
print(cm)

# Plot Confusion Matrix
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Not Spam',
'Spam'], yticklabels=['Not Spam', 'Spam'])
plt.title('Confusion Matrix for KNN')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
### Feed-Forward Neural Network (FFNN)
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.regularizers import l2
from sklearn.metrics import accuracy_score, classification_report

# Initialize TensorFlow Neural Network model with minimal complexity
ffnn_model = Sequential([
    Dense(10, activation='relu', input_shape=(10000,),
kernel_regularizer=l2(0.1)),  # Very few neurons, strong L2 regularization
    Dropout(0.8),                                                          #
Very high dropout
    Dense(1, activation='sigmoid')                                         #
Single output layer, no hidden layers
])

# Compile with extremely high learning rate
optimizer = tf.keras.optimizers.Adam(learning_rate=0.1)  # Very high learning
rate
ffnn_model.compile(optimizer=optimizer, loss='binary_crossentropy',
metrics=['accuracy'])

# Train with minimal epochs and very small batch size
ffnn_model.fit(x_train, y_train, epochs=1, batch_size=4, verbose=0,
validation_split=0.2)

# Predict
y_pred_ffnn = (ffnn_model.predict(x_test) > 0.5).astype(int).flatten()

# Calculate metrics
accuracy_ffnn = accuracy_score(y_test, y_pred_ffnn)
print("Accuracy for Feed-Forward Neural Network (FFNN):", accuracy_ffnn)

# Classification Report for FFNN Classifier
class_report_ffnn = classification_report(y_test, y_pred_ffnn)
print("Classification Report (Feed-Forward Neural Network (FFNN)):")
print(class_report_ffnn)

# Compute Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix for FFNN:")
print(cm)

# Plot Confusion Matrix
```

```
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Not Spam',
'Spam'], yticklabels=['Not Spam', 'Spam'])
plt.title('Confusion Matrix for FFNN')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# # Model saving
import pickle

# Save multinomialnb model
with open(r'Models\multinomialnb.sav', 'wb') as file:
    pickle.dump(multinomialnb, file)

# Save dt_classifier model
with open(r'Models\desicion_tree.sav', 'wb') as file:
    pickle.dump(dt_classifier, file)

# Save ada_boost model
with open(r'Models\adaboost.sav', 'wb') as file:
    pickle.dump(ada_boost, file)

# Save mlp_classifier model
with open(r'Models\mlp.sav', 'wb') as file:
    pickle.dump(mlp_classifier, file)

# Save svm_classifier model
with open(r'Models\svm.sav', 'wb') as file:
    pickle.dump(svm_classifier, file)

# Save KNN model
with open(r'Models\knn.sav', 'wb') as file:
    pickle.dump(knn_classifier, file)

# Save tensorflow neural network model
ffnn_model.save(r'Models\ffnn.h5')
```

2) app.py (Model deployment code)

```
import pandas as pd
from flask import *
import mysql.connector

db=mysql.connector.connect(host='localhost',user="root",password="",port='3306
',database='youtube')
cur=db.cursor()
```

```
app=Flask(__name__)
app.secret_key = "fghhdfgdfgrthrttgdfsadfsaffgd"

@app.route('/')
def index():
    return render_template("index.html")


@app.route('/login',methods=['POST','GET'])
def login():
    if request.method=='POST':
        useremail=request.form['useremail']
        session['useremail']=useremail
        userpassword=request.form['userpassword']
        sql="select count(*) from user where Email='%s' and 
Password='%s'"%(useremail,userpassword)
        # cur.execute(sql)
        # data=cur.fetchall()
        # db.commit()
        x=pd.read_sql_query(sql,db)
        count=x.values[0][0]

        if count==0:
            msg="user Credentials Are not valid"
            return render_template("login.html",name=msg)
        else:
            s="select * from user where Email='%s' and 
Password='%s'"%(useremail,userpassword)
            z=pd.read_sql_query(s,db)
            session['email']=useremail
            pno=str(z.values[0][4])
            name=str(z.values[0][1])
            session['pno']=pno
            session['name']=name
            return render_template("userhome.html",myname=name)
    return render_template('login.html')

@app.route('/registration',methods=["POST","GET"])
def registration():
    if request.method=='POST':
        username=request.form['username']
        useremail = request.form['useremail']
        userpassword = request.form['userpassword']
        conpassword = request.form['conpassword']
        Age = request.form['Age']

        contact = request.form['contact']
        if userpassword == conpassword:
            sql="select * from user where Email='%s' and
Password='%s'"%(useremail,userpassword)
            cur.execute(sql)
            data=cur.fetchall()
            db.commit()

            if data==[]:
                sql = "insert into
user(Name,Email,Password,Age,Mob)values(%s,%s,%s,%s,%s)"
                val=(username,useremail,userpassword,Age,contact)
                cur.execute(sql,val)
                db.commit()
                msg="Registered successfully","success"
                return render_template("login.html",msg=msg)
            else:
                msg="Details are invalid","warning"
                return render_template("registration.html",msg=msg)
        else:
            msg="Password doesn't match", "warning"
            return render_template("registration.html",msg=msg)
    return render_template('registration.html')


@app.route('/view data',methods = ["POST","GET"])
def view_data():
    df = pd.read_csv(r'dataset\YoutubeSpamMergedData.csv')
    df.head(2)
    return render_template('view data.html',col_name = df.columns,row_val =
list(df.values.tolist()))



@app.route('/model',methods = ['GET',"POST"])
def model():
    if request.method == "POST":
        algo = request.form['algo']

        if algo == 'MultinomialNB':
            algorithm = 'Multinomial Naive Bayes'
            accuracy, precision, recall, f1_score = 91.33, 91.86, 91.33, 91.33

        elif algo == 'DecisionTree':
            algorithm = 'Decision Tree Classifier'
            accuracy, precision, recall, f1_score = 94, 94, 94, 93.99

        elif algo == 'AdaBoost':
            algorithm = 'AdaBoost Classifier'
            accuracy, precision, recall, f1_score = 88.66, 89.76, 88.66, 88.51
        elif algo == 'MLPClassifier':
            algorithm = 'MLP Classifier'
            accuracy, precision, recall, f1_score = 92.33, 92.36, 92.33, 92.32

        elif algo == 'SVC':
            algorithm = 'Support Vector Classifier'
            accuracy, precision, recall, f1_score = 95, 95.08, 95, 94.98

        elif algo == 'RandomForest':
            algorithm = 'Random Forest Classifier'
            accuracy, precision, recall, f1_score = 95, 95.03, 95, 94.99

        elif algo == 'ExtraTreesClassifier':
            algorithm = 'Extra Trees Classifier'
            accuracy, precision, recall, f1_score = 95.33, 95.33, 95.33, 95.33

        return render_template('model.html', algorithm = algorithm, accuracy =
accuracy, precision = precision, recall = recall, f1_score = f1_score)
    return render_template('model.html')

from sklearn.feature_extraction.text import HashingVectorizer
hvectorizer =
HashingVectorizer(n_features=10000,norm=None,alternate_sign=False,stop_words='
english')

import pickle
with open(r"models\adaboost.sav", "rb") as file:
    model = pickle.load(file)

def prediction_func(input):
    def text_clean(CONTENT):
        # changing to lower case
        lower = CONTENT.lower()

        # Replacing the repeating pattern of &#039;
        pattern_remove = lower.replace("&#039;", "")

        # Removing all the special Characters
        special_remove = pattern_remove.replace(r'[^\w\d\s]',' ')

        # Removing all the non ASCII characters
        ascii_remove = special_remove.replace(r'[^\x00-\x7F]+',' ')

        # Removing the leading and trailing Whitespaces
        whitespace_remove = ascii_remove.replace(r'^\s+|\s+?$','')

        # Replacing multiple Spaces with Single Space
        multiw_remove = whitespace_remove.replace(r'\s+',' ')
        # Replacing Two or more dots with one
        dataframe = multiw_remove.replace(r'\.{2,}', ' ')
        return dataframe

    cleaned_text = text_clean(input)
    result = model.predict(hvectorizer.transform([cleaned_text]))

    if result == 0:
        return "This is a Not Spam Comment"
    else:
        return "This is a Spam Comment"


@app.route('/prediction', methods=['POST', 'GET'])
def prediction():
    if request.method == "POST":
        text = request.form['text']
        result = prediction_func(text)

        return render_template('prediction.html', result = result)
    return render_template('prediction.html')


if __name__=="__main__":
    app.run(debug=True)
```

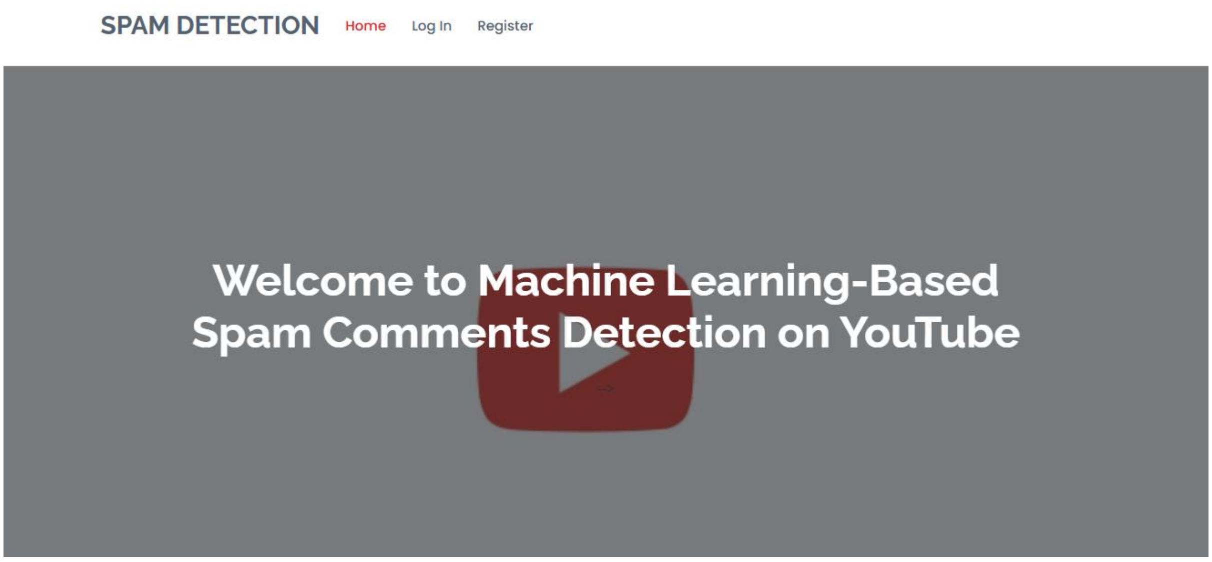
3) db.sql (MySQL database queries)

```
DROP DATABASE IF EXISTS `youtube`;
CREATE DATABASE `youtube`;
USE `youtube`;

CREATE TABLE `user` (
  `Id` int(200) NOT NULL AUTO_INCREMENT,
  `Name` varchar(200) DEFAULT NULL,
  `Email` varchar(200) DEFAULT NULL,
  `Password` varchar(200) DEFAULT NULL,
  `Age` varchar(200) DEFAULT NULL,
  `Mob` varchar(200) DEFAULT NULL,
  PRIMARY KEY (`Id`)
);
```

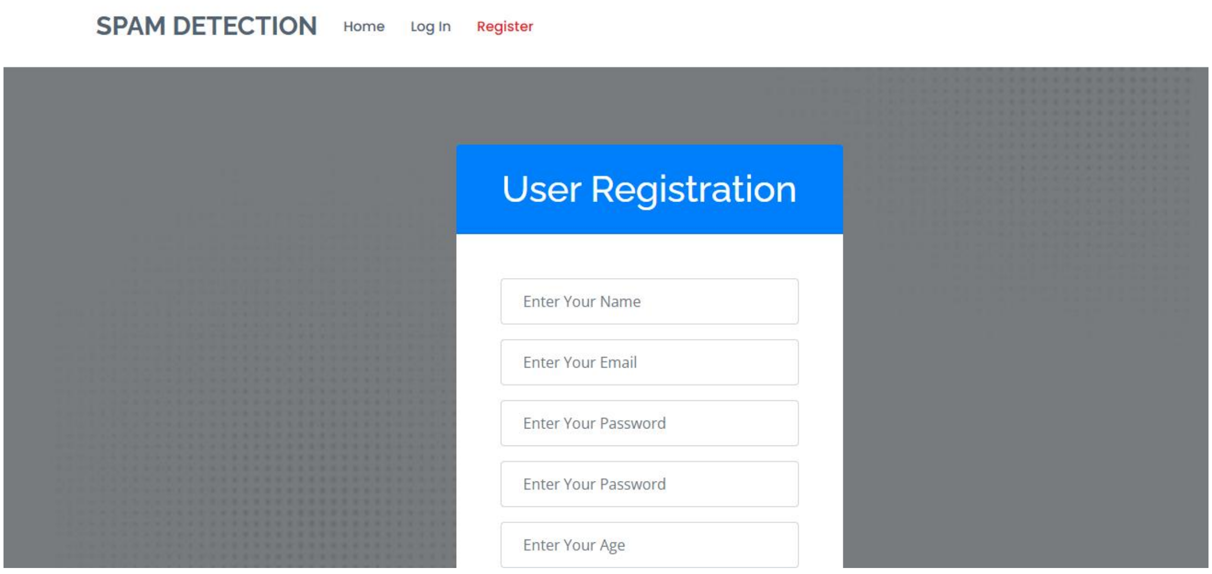
10.3 SCREEN SHOTS

- Home page: This is You tube Spam Detection home page.



Welcome to Machine Learning-Based Spam Comments Detection on YouTube

- Register Page: User need register with his/her credentials.



User Registration

Enter Your Name

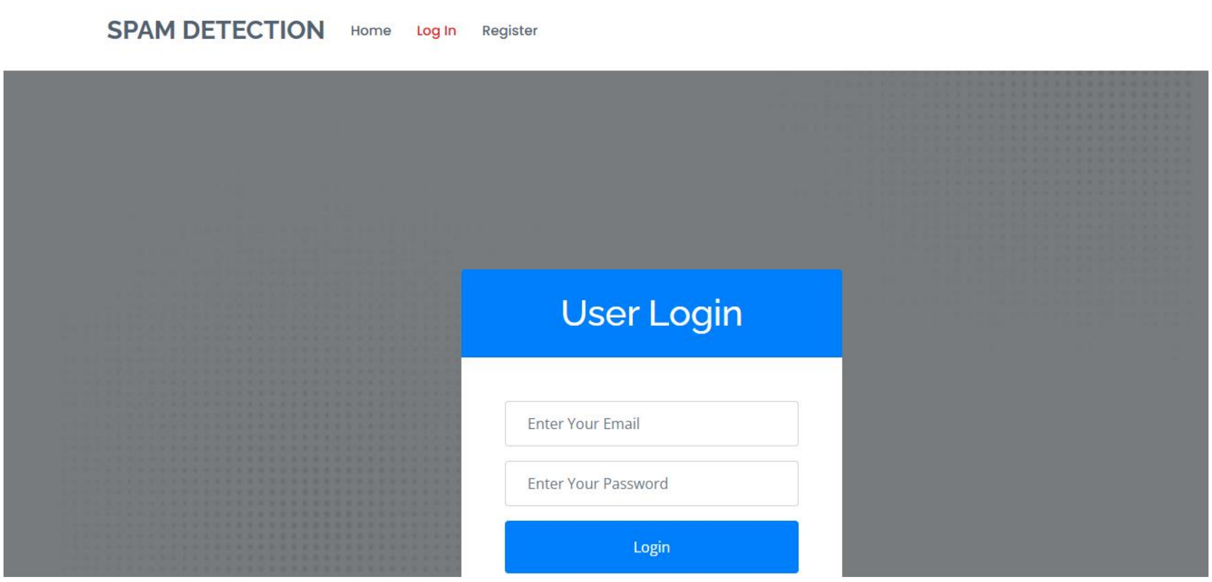
Enter Your Email

Enter Your Password

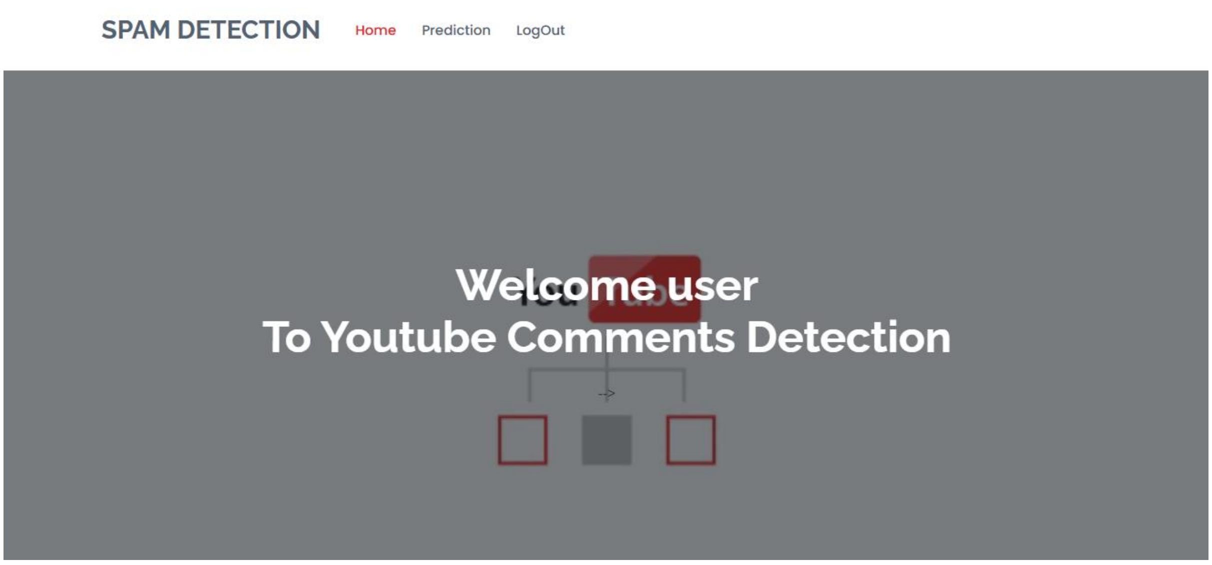
Enter Your Password

Enter Your Age

- Login page: Here the user will login with his credentials.

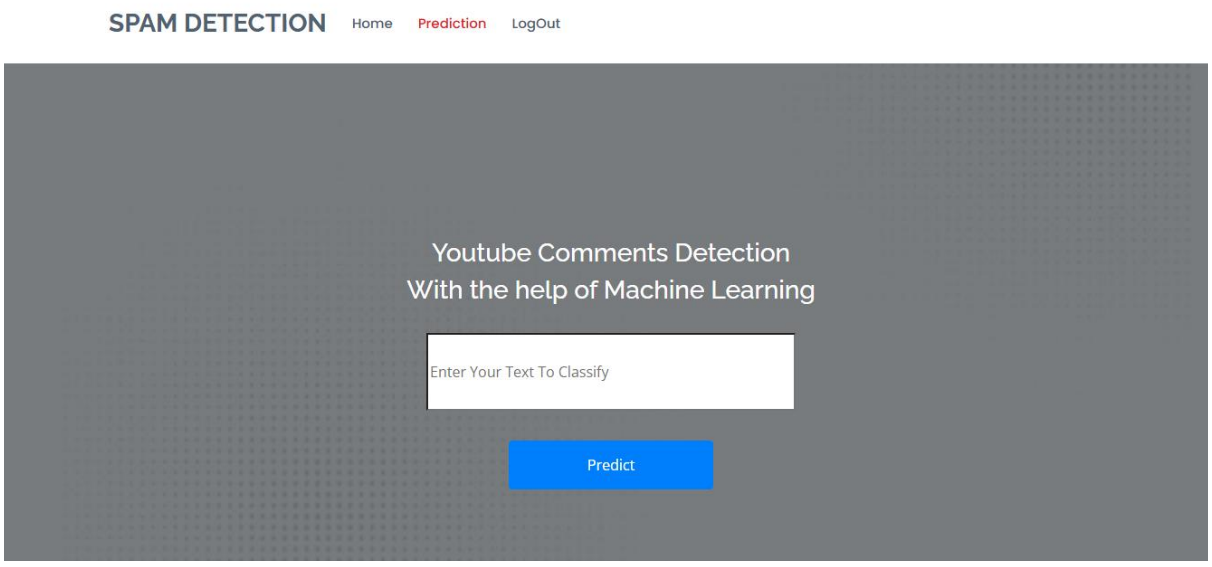


- User home page: After the user login this home page will display.

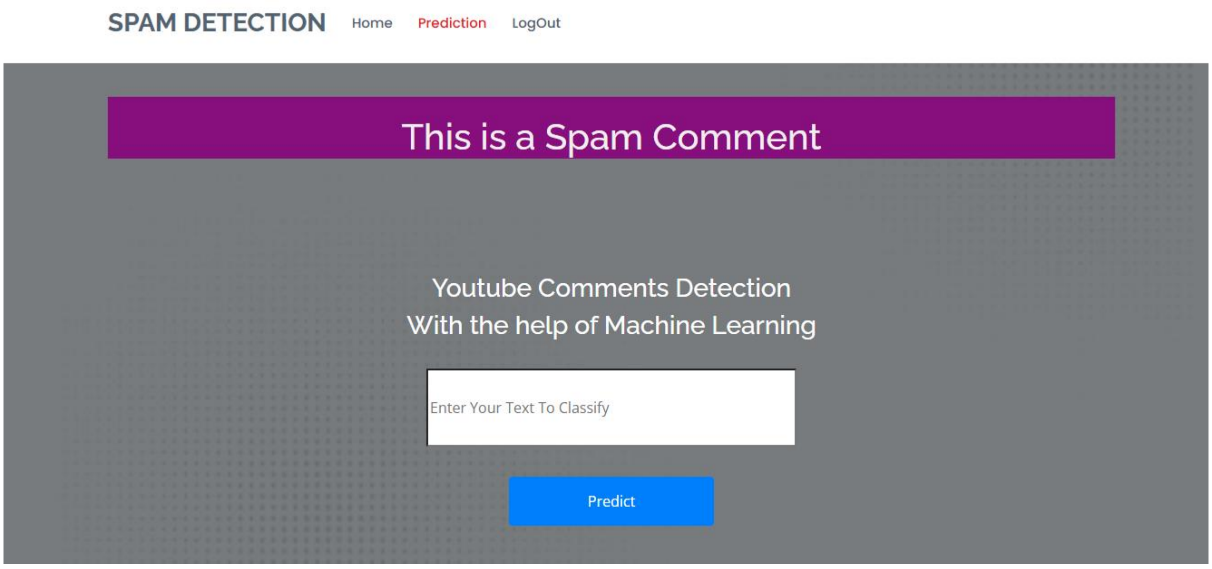


Welcome user
To Youtube Comments Detection

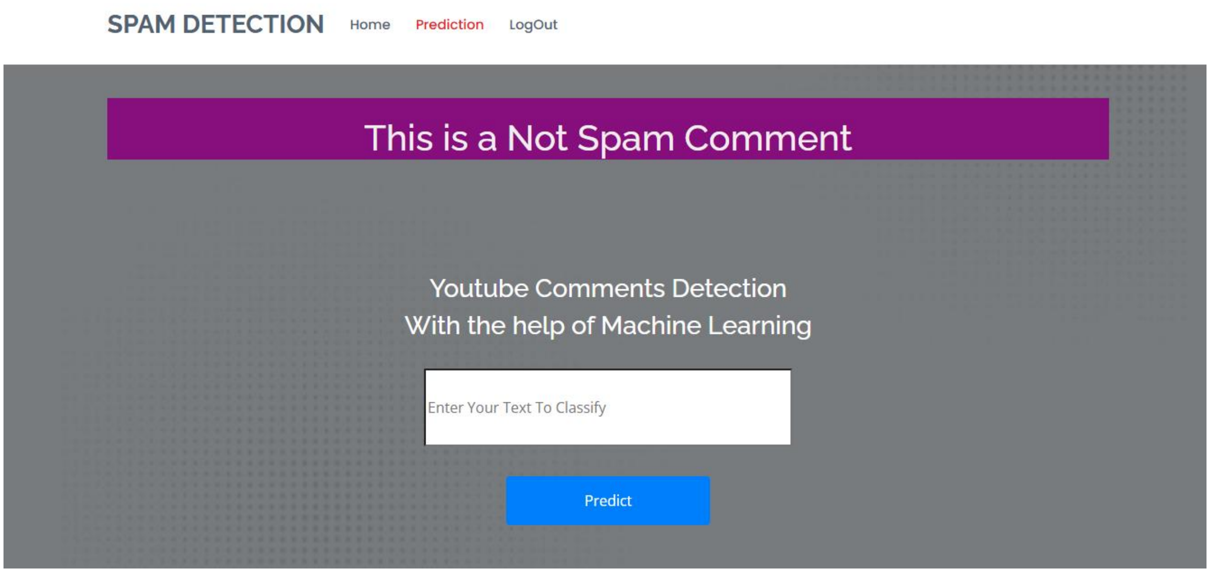
- Prediction: User need to enter input based on selecting model, it will predict whether it spam or not.


SPAM DETECTION
Home
Prediction
LogOut
Youtube Comments Detection
With the help of Machine Learning
Enter Your Text To Classify
Predict

- Result page: In this page result will display.
    - Prediction: Spam Content


SPAM DETECTION
Home
Prediction
LogOut
This is a Spam Comment
Youtube Comments Detection
With the help of Machine Learning
Enter Your Text To Classify
Predict

- Prediction: No Spam Content


SPAM DETECTION
Home
Prediction
LogOut
This is a Not Spam Comment
Youtube Comments Detection
With the help of Machine Learning
Enter Your Text To Classify
Predict

11. REFERENCES

[1] Yadav, S., & others. (2025). Spam Detection in YouTube Comments: A Machine Learning Approach. ResearchGate.
https://www.aaai.org/Papers/Workshops/1998/WS-98-05/WS98-05-007.pdf

[2] Xiao, A. S., & others. (2024). Spam Detection for YouTube Video Comments Using Machine Learning Approaches. ResearchGate.

[3] Feng, W., & others. (2016). A Support Vector Machine Based Naive Bayes Algorithm for Spam Filtering. ResearchGate.

[4] Singh, S. (2018). SMS Spam Detection Using Different ML Models: Multinomial Naive Bayes, Support Vector Machine, K-Nearest Neighbors, Random Forest, and AdaBoost. GitHub.

[5] Jain, A. (2020). Build SMS Spam Classification Model Using Naive Bayes & Random Forest. Towards Data Science.
https://www.mdpi.com/2078-2489/10/4/150

[6] Kumar, V., & others. (2024). Spam Detection on YouTube Comments Using Advanced Machine Learning Models: A Comparative Study. ResearchGate.

[7] Bassiouni, M., & others. (2022). Deep Convolutional Forest: A Dynamic Deep Ensemble Approach for Spam Detection in Text. Complex & Intelligent Systems.

[8] Sultana, A., & others. (2023). Fake News Detection Using Machine Learning Techniques. ResearchGate.

[9] Albraikan, A., & others. (2023). A Knowledge-Aware NLP-Driven Conversational Model to Detect Deceptive Contents on Social Media Posts. ScienceDirect.

[10] Zhang, L., & others. (2022). A Survey on Text Classification: From Traditional to Deep Learning. ACM Transactions on Intelligent Systems and Technology.

[11] Gupta, R., & others. (2023). Analytics of Machine Learning-Based Algorithms for Text Classification. ScienceDirect.
https://www.mdpi.com/2078-2489/10/4/150

[12] Tretyakov, K. (2006). Top 10 Data Mining Algorithms: C4.5, k-Means, SVM, Apriori, EM, PageRank, AdaBoost, kNN, Naive Bayes, and CART. IEEE International Conference on Data Mining.

[13] Wohlwend, B. (2023). Classification Algorithms: KNN, Naive Bayes, and Logistic Regression. Medium.

[14] Tonye, G. (2021). Machine Learning Confidence Scores: All You Need to Know as a

Conversation Designer. Medium.

[15] Google for Developers. (2025). Machine Learning Glossary: Confusion Matrix, Precision, Recall, F1-Score.

[16] InterviewPrep. (2023). Top 25 Classification (Machine Learning) Interview Questions and Answers.
https://link.springer.com/article/10.1007/s42979-021-00592-x

[17] Mindee. (2024). Understanding Confidence Scores in Machine Learning: A Practical Guide.

[18] Analytics Vidhya. (2024). Naive Bayes and SVM Implementation in Python.

[19] Hindawi. (2023). Analyzing Machine Learning Enabled Fake News Detection Techniques for Diversified Datasets.
https://papers.nips.cc/paper/5782-character-level-convolutional-networks-for-text-classification.pdf

[20] PNR Journal. (2023). Journal of Pharmaceutical Negative Results: Negative Results in Machine Learning Research.